View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by ScholarBank@NUS

FAULT DETECTION AND CORRECTION MODELING OF SOFTWARE SYSTEMS

WU YANPING

NATIONAL UNIVERSITY OF SINGAPORE

2008

FAULT DETECTION AND CORRECTION MODELING OF SOFTWARE SYSTEMS

WU YANPING
(B.S., USTC)

A THESIS SUBMITTED
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING
NATIONAL UNIVERSITY OF SINGAPORE

2008

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Professor Xie Min, and Dr. Ng Szu Hui. Dr. Ng and Prof. Xie has been my advisor ever since I came to National University of Singapore in 2004. During my three more years' PhD program here, Prof. Xie has been a great mentor for me, leading me into and going further in the academic field. I am very grateful for his guidance, suggestions, patience, and encouragement, which helps me to conduct my research effectively and get through some difficult conditions. This thesis would not have been possible without Prof. Xie's help. Dr. Ng Szu Hui is always available for my questions and asking for help. I have also learned a lot from Dr. Ng as her teaching assistant, both the scientific knowledge and the way to be a good teacher. Thank you very much, Dr. Ng!

I would also like to thank the other faculty members for the modules I have ever taken. Thank you, Prof. Goh, Prof. Poh, Prof. Tang, Dr. Jaruphongsa, Dr. Chai and Dr. Lee. Also, I would like to thank Ms. Lai Chun for the convenience they provided during my study and research period in our department. In addition, I would like to thank both the seniors and juniors within the batches of Prof. Xie's students, especially Dr. Hu Qingpei, Liu Jiying, Zhang Lifang, Jiang Hong, Long Quan, Zhang Haiyun, Shen Yan and Qian Yanjun. Also, thanks are due to the other student friends, especially members in the Computing Lab. I really enjoyed the time spending together with all of you!

Finally, I am grateful to my mother and my father in China for their love and support.

TABLE OF CONTENTS

SUMMARY

This thesis investigates the modeling problem of software reliability, extending traditional reliability models through relaxing some specific restrictive assumptions. Related analysis issues, especially optimal release time and optimal resource allocation, are addressed with the corresponding extended models. Centered on this line, research has been developed as follows.

Extended software reliability modeling approaches are proposed through combining both FDP (fault detection process) and FCP (fault correction process). Traditional software reliability models assume immediate fault correction. However, practical software testing process is composed of three sub-processes: fault detection, fault correction and fault introduction. We proposed the combined fault detection and correction modeling by considering various fault correction time. Our extensions are developed with both traditional NHPP and BN models, with paired NHPP and BN modeling frameworks proposed. Practical numerical application is developed for the purpose of illustration. Analysis results show the advantage of the incorporation of the fault correction process into the software reliability modeling framework. Basing on paired FDP and FCP models, time problem of optimal release is explored as well. We have further developed the software cost models based on our proposed fault detection and correction models.

Our study follows the intuitive approach of incorporating historical failure data into the frameworks of current models. Different approaches are proposed to incorporate the data collected from previous similar projects/releases. For paired FDP and FCP models, we

assume the testing and debugging environments keep stable over two consecutive projects. As a result, the fault detection and correction rates will not vibrate a lot, and then the rates estimated from previous project can be utilized in the early phase of current project. Failure data from multiple similar projects can be incorporated. Case studies conducted with two applications show the better performance of this approach in the early phase.

Besides considering the fault correction time during software testing process, we can also improve the software reliability via review and walk-through during the inspection process. For the Bayesian networks application in software reliability, we also explore the issue of software inspection effectiveness analysis. Software inspection has been broadly accepted as a cost effective approach for software defect removal during the whole software development lifecycle. To keep inspection under control, it is essential to measure its effectiveness. As human-oriented activity, inspection effectiveness is due to many uncertain factors that make this study a challenging task. Bayesian Networks are powerful for reasoning under uncertainty and have been used to describe the inspection procedure. With this framework, some further extensions are explored in this thesis. The number of remaining defects in the software is incorporated into the proposed framework, providing more information on the dynamic changing status of the inspection process. Also, a systematic approach to extract prior information is studied with a numerical example for detailed illustration.

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

FDP	Fault Detection Process
FCP	Fault Correction Process
GO-Model	Goel-Okumoto Model
NHPP	Non-homogeneous Poisson Process
SR	Software Reliability
SRGM	Software Reliability Growth Model
MLE	Maximum Likelihood Estimation
LSE	Least Square Estimate
MSE	Mean Square Error
MSE_d	Mean squares of errors of fault detection process
MSE_c	Mean squares of errors of fault correction process
RE	Relative Errors
MRE	Mean of Relative Errors
$m_d(t)$	mean value function of FDP
$m_c(t)$	mean value function of FCP
$\lambda_d(t)$	the intensity function
a	total number of detected faults
b	fault detection rate per fault
Δ_i	$i = 1,2,\cdots \overset{i.i.d.}{\sim} \Delta$, the time delay between FCP and FDP
Δ_i^+	$\{\Delta_i \mid \Delta_i > t_i - t_{i-1}\}$
Δ_i^-	$\{\Delta_i \mid \Delta_i \le t_i - t_{i-1}\}$

n_i	cumulative number of faults detected by time t_i
m_i	cumulative number of faults corrected by time t_i
$\theta_0 \in \Theta \subset R^m$	parameters in the fault detection and correction modeling
$P(n_i, m_i)$	the probability of detecting n_i faults and correcting m_i faults by time t_i
$P_d(\cdot)$	the probability distribution function for the detection process
$P_c(\cdot)$	the probability distribution function for the correction process
L	the likelihood function
Z_α	the (1-α) quantile of the standard *s*-normal distribution
RE	the predictive validity
MRE	the mean of relative errors
T	software release time
T^*	optimal release time
$E(T)$	the expected total cost of a software system at time T
c_1	the expected cost of removing a fault during the testing phase
c_2	the expected cost of removing a fault during the operation phase
c_3	the expected cost per unit time for testing
R	software reliability

Chapter 1 Introduction

Nowadays, computer systems composed of both hardware and software are widely used in everyday life in this world. As software systems play an increasingly important role in complex systems, the reliable performance of software systems becomes an important issue. Since 1970 researches have been conducted to study the reliability of the software system. Methodologies for assuring software reliability form an important part of reliability studies. With new technologies, the reliability of hardware can achieved quite a high level, while the reliability of software can still dependents greatly on human factors. As it is well known, software reliability is the application of statistical techniques to data collected during system development and operation to specify, predict, estimate, and assess the reliability of software-based systems. Since there are many human factors related operation, the reliability of software can not achieve as high level as hardware does. Thus, the reliability of software has become the focus of basic requirement for computer system. The reliability of software can get even worse with the increase of software complexity at the same time. The software crisis is often talked about when problems are involved with software products, for example, increasing development cost, lack of the ability to perform an intended task correctly, etc. The application of software systems has now crossed many different areas. Software has become an essential part of many industrial and commercial systems. Furthermore, it also plays an important role in military systems. In the high automated aviation industry, misunderstandings between computers and pilots have been implicated in several airline crashes in the past few years (Lyu, 1996). As a result, the need for reliable software has attracted great interest in both

practice and research in the software community. Therefore, developing the required techniques for software reliability engineering is a major challenge. That is the motivation for us to carry out the fault detection and correction analysis within the software system.

Lots of research in software reliability modeling has been developing for over three decades. Many models have been developed to adapt to different testing environments and under different assumptions as well (Xie, 1991; Lyu, 1996). These models provide essential tools for software reliability prediction, estimation, and assessment. These measurements are essential for the management to make decision in this phase, such as software cost analysis (Huang et al., 2003; Xie et al., 2004a), testing-resource allocation (Yamada et al., 1995; Dai et al., 2004), optimal release policy (Xie and Hong, 1999; Chang and Jeng, 2006), and fault-tolerance system analysis (Han et al., 2003; Levitin, 2005).

Those traditional software reliability models have been successfully applied in practice, and until now there are currently a number of practical papers summarizing their application experience (Musa, 1993), and providing some unified theories for software reliability models (Huang et al., 2003; Lee et al., 2004). There are many factors being considered and those traditional software reliability models are being revised based on more practical assumptions (Chang, 2001; Huang and Kuo, 2003; Pham and Zhang, 2003; Pham, 2003; Shyur, 2003; Zhang et al., 2003; Chiu et al., 2008; Lin and Huang, 2008; Kapur et al., 2008).

In software reliability literature, different authors use different synonyms referring to software reliability problems, such as fault, defect, bug, etc. A fault is always an existing part in the software and it can be removed by correcting the erroneous part of the software (Xie, 1991). Some authors use the word defect, error, bug, etc, these terminologies need to be clarified and to be unified. In this thesis, as we mainly discuss about the fault detection and correction process modeling of the software system, we unify different synonyms and use the word fault. Generally, during the software testing process, program code is executed and the erroneous outputs are identified. For each incorrect output, it can be count as a failure (Xie, 1991). Faults that caused the failure are identified and removed. Thus, the failure process during the software testing phase can be identified as a process for fault detection and correction. The reliability of the software will be increased as more and more faults are being detected and corrected. The reliability improvement phenomenon is then called reliability growth (Xie, 1991). However, the assessment of the software reliability is not easy as they are many factors lead to failure. The level of the reliability is usually estimated by using some appropriate models applied to the empirical data from the software failure history.

1.1 Fault Detection and Correction Modeling

Software reliability modeling plays a critical role in software development, particularly during the software testing stage. In the last few decades, generalizations and extensions of software reliability growth models (SRGMs) have continued to attract researchers in the field. The software reliability models can be categorized into two groups: analytical

software reliability models and data-driven software reliability models (Musa et al., 1987; Xie, 1991; Lyu, 1996; Pham, 2000). Bothe analytical and data-driven modeling approaches have their model assumptions which can be exposed by dividing the testing process into three sub-processes: fault-detection, fault correction, and fault introduction. Analytical models assume perfect and immediate fault correction. Data-driven models only analyze the historical data from the fault detection process, ignoring the collected fault correction data. As a result, fault correction is not incorporated for both approaches.

According to different modeling techniques, these models can also be grouped into NHPP (non-homogeneous Poisson process) models, Markov models, and Bayesian models. Among these three models, NHPP models are applied broadly for their flexibility and simplicity, and Bayesian models are mostly developed from the corresponding Markov and NHPP models. Analytical software reliability models describe the software failure behavior during the software testing process and model the process as a stochastic process, while data-driven models focus on the failure data generated through the software testing process and model the software reliability prediction as a time-series analysis problem.

However, there are some restrictive assumptions for those general models. The reason for this is probably that the assumptions made for each model are correct or are good approximations of the reality just in some situations. Those restrictive assumptions are not compatible with the practical software testing/developing environments. These assumptions might not be realistic in practice or too complicated to be realized.

One thing of great interest and attracts attention is that it is not realistic and practical to ignore the fault correction in software reliability modeling. Although there are many research papers on software reliability modeling, few of them address the realistic time delays between fault detection and fault correction processes. Most of the models consider only software fault detection process in the testing stage, assuming perfect and immediate fault correction with no debugging time. While in fact, in reality, each detected fault is reported, diagnosed, corrected, and then verified. The time between detection and correction should not be neglected in practical software testing process (Zhang and Pham, 1998).

Unlike fault introduction, the fault correction data can be extracted from related historical reports. With more information of fault correction data, software reliability models considering both fault detection and correction can be developed. Recently, more and more researchers emphasized the great importance of the fault correction modeling (Schneidewind, 1975; Xie and Zhao, 1992; Schneidewind, 2001; Schneidewind, 2003; Stutzke and Smidts, 2001; Bustamantea and Bustamante, 2003; Zhang et al., 2003; Hu et al., 2007). However, due to lack of actual data, no systematic work has been carried further in modeling the fault detection and correction processes together based on NHPP models.

Fault correction is a difficult and time-consuming exercise. When the performance of fault detection and fault correction are to be evaluated from test data to measure the software reliability, the evaluation method is usually to construct a reliability model.

These models use empirical data and assumptions about the software development process, and they usually result in estimation of model parameters and prediction of future failures. As a result, combined fault detection and correction modeling could present more practical models for software testing process, and it could give more accurate reliability prediction and trend analysis, which could provide crucial information for decision making and reliability engineering for most projects. Therefore, research has been focused on extending the modeling by relaxing some restrictive assumptions so as to adapt to flexible software environments. That is the motivation for us to make some further development based on traditional software reliability growth models.

Besides realistic modeling, the problem of accurately estimating software faults remains a difficult one. Fitting a proposed model to the actual data of faults detection and correction involves estimating the model's parameters from the real test data sets. Once able to estimate those parameters, we can give accurate predictions to the future behavior of the fault detection and correction process, which will help software managers to allocate testing resources and study the software release problems.

Parameter estimation method is also addressed in this thesis. For traditional software reliability models, Least Square (LS) estimation method has been applied in most studies to estimate the complex fault detection modeling parameters (Xie et al., 2007; Inoue and Yamada, 2006; Zhao et al., 2006; Jiang and Xu, 2007). However, it is well accepted that the Maximum likelihood estimation (MLE) is one of the most popular estimation techniques with many desirable properties, such as asymptotic normality, admissibility,

robustness and consistency, and it is quite straightforward and has been widely used to estimate the parameters for SRGMs (Inoue and Yamada, 2004; Zou, 2003; Musa et al., 1987; Schneidewind, 1993; Xie, 1991). Maximum likelihood (ML) parameters are estimated by solving a set of simultaneous equations and then the confidence interval of those parameters can be easily derived. Up to now, no MLE method has been applied in the existing studies to estimate parameters in the fault detection and correction process. In this thesis, we take into account the time dependency and consider the issue of applying the ML estimation method to the combined FDP and FCP from both a theoretical and an experimental perspective.

Once the parameters are estimated, accurate predictions of the future failure behavior can be made. In addition to predicting the number of faults remaining in software, other process characteristics can also be estimated. With the realistic consideration of the time dependencies, more accurate estimations and decisions can be made on managing project resources. A direct and useful application of this combined time dependent model is in the optimal release time determination problem.

In the first part of this thesis, a systematic study on the fault detection and correction process is carried out, a framework is proposed to incorporate the time dependencies between the fault detection and fault correction processes with the emphasis on the fault correction process. Various fault correction models are proposed considering different forms of the time delay.

1.2 Inspection Effectiveness Model with Bayesian Networks

The study on the fault detection and correction process develops a method to help software managers to make a decision of when to release the software so as to receive a high reliability and satisfied quality. However, since the cost for the fault detection and correction during the software testing phase is considerably high, another question comes into our consideration. That is, how to remove as many software faults as possible while keeping the debugging cost relatively low within the software development lifecycle.

Generally, the longer a defect remains in a product, the more costly it is to remove it. Research showed that a high proportion of software errors were introduced at the start of the development lifecycle during the requirement phase (Delic et al., 1995). In addition, further faults may be introduced by the fault debugging. Considering this point of view, it is necessary to remove faults as early as possible so as to save money and energy within the development lifecycle. Except for testing, the only other widely applicable technique for detecting and eliminating software defects is to review and walkthrough during the inspection process.

Software inspection is a systematic technique to examine any software artifact for defect detection and removal. It has been broadly accepted as a cost effective approach for software defect removal during the whole software development lifecycle. It is accepted that inspection can detect and eliminate faults more cheaply than testing; the inspection method can be used to improve productivity and to shorten development schedules. In general, they can reduce cost and schedule of testing. Inspection process is considered

important, and inspection effectiveness is considered as an important criterion to judge the inspection performance. That is the motivation for us to construct models to measure the inspection effectiveness.

To keep inspection under control, it is essential to measure its effectiveness and many different attempts have been made to measure software inspection effectiveness. With this measurement, we can develop relevant decision-making, such as when to stop testing. Starting from this point, we propose a systematic method to analyze the inspection effectiveness so as to find out factors that can improve the inspection performance, that is, to improve the efficiency of detecting and eliminating software defects. A Bayesian network (BN) model is proposed to describe the interdependencies within the inspection structure and the contribution of each factor to the overall belief on inspection effectiveness, and a systematic approach is developed to extract knowledge from experts.

As human-oriented activity, inspection effectiveness is due to many uncertain factors, which makes such a study a challenging task. As we have known, Bayesian networks modeling is a powerful approach for the reasoning under uncertainty and it can describe the inspection procedure well. Based on a Bayesian networks model, extensions will be explored in several directions, and software inspection can be modeled as a dynamic process and the belief on effectiveness will be updated with new information collected. Systematic approach to extract knowledge from experts can be explored in case of introducing more uncertainty and possible inconsistency into the modeling framework.

In the second part of this thesis, some extensions have been explored modeling the inspection effectiveness with the Bayesian network framework developed in Cockram (2001). Specifically, the number of remaining defects in the software is proposed to be incorporated into the framework, with expectation to provide more information on the dynamic changing status of the software. Also, considering the learning process usually happening in software development, the dynamic evolution of inspector's experience with the advance of inspection is studied. In addition, a different approach is adopted to elicit the prior belief of related probability distributions for the network. Specially, sensitivity analysis is developed with the model to locate the important factors to inspection effectiveness.

1.3 Research Objective and Scope

The purpose of this thesis is to develop comprehensive and practical models to measure software reliability, providing more accurate information for management to make cost-effective decisions. Specifically, traditional software reliability models, both NHPP and BN, will be extended through modeling both the fault detection process and the fault correction process. Also, Bayesian networks will be used to measure the effectiveness of the software inspection, a reliability related measurement in the very early phase of software development.

Extensions on current NHPP models will generalize the time-delayed relationship between the fault detection and correction processes with a general framework. The inter-

relationship between fault detection and correction will be incorporated as well with no restrictive assumptions. For both kinds of models, software testing will be described more practically. As a result, more accurate software reliability predictions would be available to help software project managers to make decisions in activities such as cost estimation, stopping-point determination, and resource allocation.

Clearly, more data is needed than the traditional modeling frameworks. This requirement on data is usually not a problem with modern software companies, as they have plenty of historical data stored in their databases. However, few data is available in published works. Then both simulated and field data is used to illustrate the proposed approach.

The remainder of this thesis is organized as follows. In chapter 2 we provide the general background of basic software reliability models and some related software reliability analysis topics. In chapter 3 the systematic paired analytical FDP and FCP models are proposed and the related reliability analysis problem is explored there. In chapter 4 parameter estimation methods are discussed and maximum likelihood estimates of combined models are derived from an explicit likelihood formula under various time delay assumptions. In chapter 5, various characteristics of the combined model, like the predictive capability, are also analyzed and compared with the traditional least squares estimation method. Since no single comparison is adequate to determine the method with better prediction performance, a Monte Carlo simulation analysis is carried out as well. In chapter 6 we study a direct and useful application of the proposed model and estimation method to the classical optimal release time problem faced by software decision makers.

Comprehensive comparisons among various software cost models are conducted. The results illustrate the effect of time delay on the optimal release policy and the overall software development cost. In chapter 7, a revised BN model is given using NETICA software to measure the inspection effectiveness. Sensitivity analysis is carried out to identify the uncertain factors that have the largest impact on the software inspection process. Since the initialization of the BN model requires establishing the prior belief of the conditional probability distribution of intermediate variables and the prior belief of the probability distribution of the root parent nodes, two methods are proposed to obtain those prior probabilities. The first method is given through calculating the pair-wise comparison matrix using EXPERT CHOICE software. The second method is given using maximum likelihood estimation method to find out the distribution for the normalized data value and finally give the a-priori conditional probability table. The proposed method can help maximizing the inspection effectiveness, improving the efficiency of removing faults as early as possible, and finally improving the software quality even before the software testing phase begins. Chapter 8 concludes current research work and discusses some further research topics.

Chapter 2 Literature review

2.1 Software Reliability Models

Software reliability is one of a number of aspects of computer software which can be taken into consideration when determining the quality of the software. Building good reliability models is one of the key problems in the field of software reliability. A good software reliability model should give good predictions of future failure behavior, compute useful quantities and be widely applicable. Therefore, a very important goal of current software reliability research is to develop general prediction models. Existing models typically reply on assumptions about development environments, the nature of software failures and the probability of individual failure occurrences. Thus each model can be shown to perform well with a specific failure data set, but no model appears to perform well for all cases.

Generally, software reliability growth models (SRGMs) are composed of both analytical and data-driven models (Xie, 1991). Analytical SRGMs have three major sub-categories: non-homogenous Poisson process (NHPP) models, Markov models, and Bayesian models. A stochastic process is usually incorporated in the description of the failure phenomenon, such as the Markov process assumption and non-homogeneous Poisson process which are widely used. They are constructed by analyzing the dynamics of the software failure process, and their applications are developed by fitting them against software failure data.

Some other models deal mainly with the inference problems based on the failure data and these models include Bayesian models and other statistical methods.

Software reliability, defined as the probability of failure-free software operation for a specified period of time in a specified environment (Lyu, 1996), is supposed to be a good measurement to quantify software failures. Lots of software reliability growth models (SRGMs) have been proposed to measure the software failure process successfully (Teng and Pham, 2004, Huang et al., 2003; Tamura and Yamada, 2006; Xie and Yang, 2003; Shyur, 2003; Chatterjee, 2004), among them some are based on non-homogeneous Poisson process (NHPP) (Musa et al., 1987; Xie, 1991; Lyu, 1996; Pham, 2000).

In the course of development of software reliability research, many models have been built to predict future failures. Software failure dependencies are being analyzed (Dai et al., 2004, 2005; Levitin and Xie, 2006); software cost models and optimal release policies are being proposed (Xie et al., 2004a); the reliability of fault tolerant software is also analyzed (Levitin et al., 2007). Software grid service reliability is also considered (Dai et al., 2005). Some of the models are described as Non-homogeneous Poisson Process (NHPP) models, because the mean value function *m(t)* represents the cumulative number of faults exposed up to time t. in practice, many of the NHPP models are proved to be effective only in a particular environment.

Traditional SRGMs only consider the fault detection process assuming perfect and immediate fault correction. Software fault-detection process *N(t)* is usually assumed to

follow a NHPP, in which the intensity function $\lambda_d(t)$ is time-dependent. Given $\lambda_d(t)$, the mean value function (MVF) $m_d(t)$ satisfies

$$m_d(t) = \int_0^t \lambda_d(s)ds \tag{2.1}$$

The mean value function $m_d(t)$ is the characteristic of the NHPP model. Generally, different fault detection models can be obtained by using different non-decreasing functions $m_d(t)$.

There are four classical NHPP models as follows.

2.1.1 Goel-Okumoto Model

The GO-model (Goel and Okumoto, 1979) is one of the most influential NHPP software reliability models. The mean value function is given as

$$m_d(t) = a \cdot (1 - e^{-bt}),\ a, b > 0 \tag{2.2}$$

where a is the number of faults that can be detected by the testing process, and b can be interpreted as the failure occurrence rate per fault.

2.1.2 Duane Model

The Duane model which is also referred to as the Weibull process model assumes that the mean value function satisfies

$$m(t) = at^b \tag{2.3}$$

In the above, the parameters can be estimated by using collected failure data.

One of the most important advantages of the Duane reliability growth model is that if we plot the cumulative number of failure versus the cumulative testing time on a log-log-scaled paper, the plotted points tends to be close to a straight line if the model is valid. As pointed out by Xie (1991), some of the disadvantages of the Duane model are that it gives an infinite ROCOF (Rate Of oCcurrence Of Failures) at time zero and it gives zero ROCOF at time infinity. Littlewood (1984) then proposed a modified version of the Duane model.

2.1.3 Yamada Delayed S-shaped Model

The Yamada delayed S-shaped (DSS) model is an S-shaped curve for the cumulative number of detected faults. The failure rate initially increases and later decreases. Yamada assumed that the fault detection rate was a time-dependent function described by an S-shaped curve because the testers' skills would gradually improve as time went by (Xie,

1991). It is used to model the delayed reporting phenomenon for fault detection. The mean value function is given as

$$m_d(t) = a \cdot [1 - (1 + bt)e^{-bt}],\ a, b > 0 \qquad (2.4)$$

with parameter *a* denoting the number of faults to be detected, and *b* corresponding to a fault detection rate.

2.1.4 K-stage Erlangian (gamma) Growth Curve Model (k=3)

The K-stage Erlangian growth curve model, usually called the K-Model, was applied by Khoshgoftaar (1988). He observed that the Goel and Okumoto model and the S-shaped model could be described as special cases of a Gamma function. The mean value function with the value of K equal to 3 is:

$$m(t) = a\left(1 - \left(1 + bt + \frac{(bt)^2}{2}\right)e^{-bt}\right) \qquad (2.5)$$

Two special cases of the K-Model are K=1 and K=2, where the K-Model reduces to the G-O model and the S-shaped model, respectively. Usually the K-Model is studied at the case where K=3.

In these NHPP models as illustrated above, usually parameter *a* represents the mean number of software failures that will eventually be detected, and parameter *b* represents the probability that a failure is detected in a constant period. Mainly there are two classes of $m_d(t)$ used to describe different fault detection processes: concave and S-shaped models. Concave $m_d(t)$ describes the fault detection process with exponential decreasing intensity. Differently, S-shaped $m_d(t)$ describes fault detection process with increasing-then-decreasing intensity, which can be interpreted as a learning process.

To highlight the idea and approach in our study, we propose our fault detection and correction model based on the G-O model as an example, although there are many other classical SRGMs based on NHPP that can be used like the Yamada exponential model, the Yamada Rayleigh model.

Unified theories have been discussed for SRGM models (Huang et al., 2003; Sharma and Trivedi, 2007). Various factors are combined to software reliability models (Chang, 2001; Pham, 2003; Shyur, 2003; Zhao et al., 2006; Gokhale et al., 2006; Jain and Maheshwari, 2006; Huang et al., 2007). Model applications and performance analysis are carried out as well (Satoh and Yamada, 2001; Teng and Pham, 2004; Keiller and Mazzuchi, 2002; Satoh and Yamada, 2002; Nahas and Nourelfath, 2005). An overwhelming majority of publications on NHPP considers just two monotonic forms of the NHPP's rate of occurrence of failures (ROCOF): the log-linear model and the power law model (Krivtosov, 2007). Software prediction is also studied widely in current software reliability research (Li et al., 2007; Madsen et al., 2006).

2.2 Parameter Estimation

The NHPP model is a very important class of software reliability models and is widely used in software engineering. NHPPs are characterized by their intensity functions. The parametric statistical methods are often applied to estimate or to test the unknown reliability models. Maximum likelihood Estimation (MLE) method has been widely analyzed in current research. Weighted likelihood function has been proposed addressing the problem of estimating the parameter of an exponential distribution (Ahmed et al., 2005). A number of studies have been carried out to study the properties of Maximum Likelihood Estimation (Bottai, 2003; Burdick et al., 2006; You and Zhou, 2006; Zhao et al., 2006; Karlis and Meligkotsidou, 2006). Other parameter estimation methods are also discussed, such as Bayesian method (Goldstein and Bedford, 2006) and Markov Chain Monte Carlo (MCMC) method (Pang et al., 2007).

2.3 Optimal Release Policy

As software systems become more and more complex, they are prone to having more and more faults inside. Increased software system complexity challenges software mangers and testers to maintain quality control over the development process with effective and efficient test plans. While exhaustive testing of software can ensure the deployment of high quality software, exhaustive testing is never practical due to the significant costs of running many test cases. In contrast, if the software is tested inadequately, then failures during the actual deployment of the software can lead to significant expenses involved in fixing the software, loss of goodwill, and potential legal liabilities. What is needed is an

optimal level of testing that balances the risks of failures with the costs incurred while testing the software to meet software reliability requirements. With different software reliability models combined with different release criteria, there are many papers dealing with this topic (Ross, 1985; Dalal, 1988; Littlewood, 1997; Kimura et al., 1999; Xie and Hong, 1999; Zhang and Pham, 1998; Dai et al., 2004; Xie, 2004a; Huang, 2005a).

One of the challenging problems for software companies is to find the optimal time of release of the software so as to minimize the total cost expended on testing and potential penalty cost due to unresolved faults. If the software is for a safety critical system, then the software release time becomes more important. Bhaskar and Kumar (2006) developed a total cost model based on criticality of the fault and cost of its occurrence during different phases of development for N-version programming scheme, a popular fault-tolerant architecture. Boudali and Dugan (2006) presented a continuous-time Bayesian network (CTBN) framework for dynamic systems reliability modeling and analysis. Chang and Jeng (2006) investigated stopping rules for software testing and proposed two stopping rules from the aspect of software reliability testing based on the impartial reliability model.

The overall lifecycle cost associated with product failures exceeds 10% of yearly corporations' turnover. A major factor contributing to the loss is ineffective performance of software and systems verification, validation and testing (VVT). Engel and Last (2006) then proposed a set of quantitative probabilistic models for estimating costs and risks stemming from carrying out any given VVT strategy. Fenton et al. (2007) described a

more general approach that allowed causal models to be applied to any lifecycle. For projects within the range of the models, defect predictions are very accurate. This approach enabled decision-makers to reason in a way that was not possible with regression-based models.

Pham and Wang (2001) modeled software reliability and testing costs using a quasi-renewal process. Xie and Yang (2003) extended a commonly used cost model to the case of imperfect debugging, which means that faults are not immediately corrected and more time are needed to locate and correct it. Xie et al. (2004a) presented a general cost model and a solution algorithm for the determination of the optimal number of hosts and optimal system debugging time. Huang (2005b) proposed a software cost model that could be used to formulate realistic total software cost projects and discussed the optimal release policy based on cost and reliability considering testing effort and efficiency. Teng and Pham (2004) first incorporated the random field environmental factor into the cost model.

The determination of the optimal release time for a new piece of software is of primary importance in the process of software development. Boland and Chuiv (2007) studied a model where initially there were N faults in the software, but where the probability of a perfect repair of a fault when found is p (in general repair is not perfect). They investigated various cost models for the situation and gave some insight into how the optimal release times and costs for the software vary with the failure detection rate and p.

2.4 Models to Measure Inspection Process

Software inspection is 'a well-structured technique that originally began on hardware logic and moved to design and code, test plans and documentation with the intended purpose of effectively and efficiently identifying defects early in the development process' (Fagan, 1976, 1986). It has been generally accepted in software development as a cost-effective approach for quality improvement through defect removal (Aurum et al., 2002). Such a static verification technique was first introduced in Fagan (1976), and has been studied and applied extensively with a variety of applications (Kelly and Shepard, 2004b; Miller and Yin, 2004). Zhao et al. (2007) developed a model to evaluate the reliability and optimize the inspection schedule for a multi-defect component.

Software inspection process is a complicated process with many uncertain factors. This process can be characterized by different objectives, participants, preparation, participants' roles, meeting duration, work product size, work maturity, output products, and the process discipline (Aurum et al., 2002). With these basic elements, different inspection processes have been introduced, such as active design review, two person inspection, N-fold inspection, phased inspection, etc. To measure the effectiveness of software inspection, the relationships of all the required variables should be addressed.

There have been many different attempts to measure software inspection effectiveness. Some works suggest using the already detected defects to calculate the measurement, i.e., defect density (Porter et al., 1997; Perry et al., 2002). Also, the status of remaining defects is proposed to be another measurement through both objective and subjective

approaches (Biffl, 2003), and Capture-recapture is a well studied approach to develop related estimation (Emam and Laitenberger, 2001; Petersson et al., 2004). As pointed out by Stringfellow (2002), the pre-screening method has a greater impact on components with few defects. One way to compensate for that problem is to look at estimators that tend to under-estimate. If overlap is reduced due to pre-screening, estimates will be higher. Estimators that tend to under-estimate will compensate for defect scrubbing.

It should be noted that the experience-based method takes scrubbing into account. Experience-based models adjust to the data. If the scrubbing is done in a similar way for all releases, the estimates should be trustworthy. However, it is criticized with the extra cost and difficulties added in defect implantation, and some alternatives are developed through the time series trend or subjective judgments on the collected data (Amasaki et al., 2005; Yin et al., 2004).

Unfortunately, these natural but simplistic measurement definitions regard software inspection as a mechanical process. There is no unified inspection structure and there are many factors contributing to its effectiveness for each specific procedure (Biffl and Halling, 2003; Briand et al., 2004). Many of these factors are highly dependent on the experience of individual inspectors and introduce great uncertainty into this process (Kelly and Shepard, 2004a; Perry et al., 2002).

2.4.1 The Importance of Measuring Inspection Process

Delic et al. (1995) found that some 70% of software faults in mission-critical space systems were due to errors introduced during requirement phase. In addition, the re-work of the previous development stages was often at considerable expense and consequent re-testing, and further faults may be introduced by the re-work. Remus and Ziles (1979) provided a simple model of error removal and integrity progression using the reliability figures from similar types of software, showing another way to reduce the remaining faults number, that is, to find as many faults as possible during the inspection process so as to improve the quality of software itself.

As the use of software products in today's world has increased dramatically making quality an important aspect of software development, there is a continuous need to develop processes to control and increase software quality. As software code inspection is one way to pursue this goal, Vreede et al. (2006) presented a collaborative code inspection process that was designed during an action research study using collaboration engineering principles and techniques. Results showed that regardless of the implementation, the process was found to be successful in uncovering many major, minor, and false-positive defects in inspected piece of code.

Along with improved quality, substantial productivity gains have also been reported. Such gains are possible for two reasons. First, the longer a defect remains in a product, the more costly it is to remove it. Second, except for reviews and walkthroughs, the only other widely applicable technique for detecting and eliminating software defects is testing.

If inspections can detect and eliminate faults more cheaply than testing, they can be used to improve productivity and to shorten development schedules.

The above shows that the inspection is very important before we begin modeling the fault detection and correction during the testing phase. That motivates us to find a way to measure the inspection effectiveness and to find out factors that can influence the inspection effectiveness. By changing those influential factors, we can improve the efficiency of detecting and eliminating software defects at the early stage of software development, therefore, help saving lots of money and energy during the testing phase.

2.4.2 A Brief Review of Software Inspection Process

It has been widely accepted that software inspection is a cost-effective approach for quality improvement through defect removal (Aurum et al., 2002). Such a static verification technique is originally introduced in Fagan (1976), and has been studied and applied extensively with many varieties (Kelly and Shepard, 2004b, Miller and Yin, 2004). Fagan (1986) described a fishbone diagram of the causal influence for the quality of software inspection, which showed the influences on the quality of inspection processes. Cockram (2001) redrawn Fagan's diagram to give an indication of the type of attributes that influenced the effectiveness of the inspection. Aurum et al. (2005) investigated the inspection effectiveness by altering some of the inspection attributes, such as the environmental context, document type and reading technique. Freimut et al. (2005) proposed a model to measure inspection cost-effectiveness and a method to

determine the cost-effectiveness by combining project data and expert opinion. Generally speaking, software inspection is a systematic technique to examine any software artifact for defect detection and removal, and can be applied to the early phase in software development.

However, software inspection process is flexible and complex. There is no unified inspection structure and there are many factors contributing to its effectiveness for each specific procedure (Biffl and Halling, 2003; Briand et al., 2004). Many of these factors are highly dependent on the experiences of individual inspectors, introducing great uncertainty into this process (Kelly and Shepard, 2004a; Perry et al., 2002). Bayesian network widely known as a powerful approach to model under uncertainty is then considered to help modeling the inspection process.

2.4.3 A Brief Introduction of Bayesian Network Models

Bayesian network (Pearl, 1986) is a directed acyclic probability graph, connecting the relative variables with arcs, and this kind of connection expresses the conditional dependence between the variables. The influence is not necessarily linear; in general if one node can take n values and the other m values, the influence of one mode on the other is a $n \times m$ matrix. Experience is used to provide a priori probability values for each node matrix. Therefore, Bayesian network is well-known as a powerful approach for reasoning under uncertainty.

To construct the Bayesian network requires three types of knowledge:

1) The structure of the network showing the node dependencies.
2) A matrix giving the conditional probability distribution for each link.
3) The structure of the network is acyclic.

Besides the Bayesian network construction, the Bayesian network can also include nodes being set to a pre-defined value (evidence nodes), which falls under probabilistic inference. Bayesian network inference is NP-hard for a general network which warrants the use of BN software such as NETICA.

The application of Bayesian networks can provide a means of initializing the model from inspectors' experience, with the model having the ability to learn and optimize its performance from the results of inspections. One of the basic assumptions in Bayesian inspection models is that some prior knowledge is given about the number of defects in a certain product or software system. The prior knowledge could be often described as a probability distribution. Chun and Sumichrast (2006) proposed three conditions that should be put forth as desirable properties for a prior probability distribution of the number of defects in the product. Various prior probability distributions were reviewed and tested if they met those conditions. The negative binomial distribution was found to be the only one that satisfied all the desirable conditions. With the negative binomial prior, the effects of various parameters were analyzed on the Bayesian estimate of the number of undetected errors still remaining in the product.

Ganssle (2001) did a survey showing some striking examples of the value of code inspections:

1) IBM managed to remove 82% of all defects before testing even began.
2) ATandT found inspections led to 14% increase in productivity and 10-fold increase in quality.
3) HP found that 80% of the errors detected during inspections were unlikely to be caught by testing.
4) HP, Shell Research, Bell Northern, and ATandT all found inspections 20-30 times more efficient than testing in detecting errors.

Over the last decades, Bayesian networks (BN) have become a popular tool for modeling many kinds of statistical problems. Langseth and Portinale (2007) discussed the properties of the modeling framework that make BNs particularly well suited for reliability applications, and point to ongoing research that is relevant for practitioners in reliability. Melo and Sanchez (2006) pointed out that Bayesian networks have been applied toe deal with uncertainties in software development recently.

Based on the diagram proposed by Fagan (1986), and the one revised by Cockram (2001), we can provide a predictive model of the effectiveness of software inspections using Bayesian network modeling.

Chapter 3 Modeling of the fault detection and correction process

Software reliability modeling is to describe fault-related behaviors of software testing process, which generally includes fault detection, correction and sometimes fault introduction. In this chapter, our research is mainly based on traditional SRGMs, and the aim is to further develop models with more realistic assumptions. Traditional SRGMs consider the fault detection process only, usually assuming that the fault detected is corrected immediately and perfectly, while in reality, it is not always that case. Imperfect correction issue has been studied comprehensively (Xie and Yang, 2003; Bhaskar and Kumar, 2006). However, relatively less research has been carried out to incorporate fault correction process into software reliability models. In fact, the time needed for fault correction can not be neglected in software testing practice. For each detected fault, it has to be reported, diagnosed, removed and verified before it could be noted as corrected. Furthermore, the fault correction time is an important factor for some critical decision analysis (Stutzke and Smidts, 2001; Zhang et al., 2003). As a consequence, combined fault detection and correction modeling could present more practical models for software testing process, with better assistance to related decision-making activities.

Schneidweind (1975) first proposed the idea of modeling the fault correction process, in which, the fault correction process is modeled as a separate process following the fault detection process with a constant time lag. It was later highlighted in Xie and Zhao (1992) where a time-dependent delay function was proposed. In Schneidewind (2001), the time delay was assumed to be an exponentially distributed random variable. However, due to

the lack of actual data showing the fault detection and correction processes, little real progress has been made.

In this chapter, a systematic study on the fault detection and correction processes is carried out. We propose new models by considering the time delay, that is, the time spent to correct the detected fault. We consider there is a time delay between fault correction and fault detection; therefore, the fault correction process can be modeled as a delayed fault detection process with random or deterministic delay. An actual data set is used to illustrate the modeling framework and reliability analysis procedure. Below we propose different models by presenting different forms of the time delay between these two processes. To highlight our idea and approach, we are using G-O model for illustrative purpose. Similar approach can be carried out based on other software reliability models as well.

3.1 The Modeling Framework of FDP and FCP

When information about the fault detection process and the fault correction process are all available, the fault correction process can be modeled as a process separate from fault detection. They can then be analyzed in a way similar to that of traditional NHPP SRGMs reviewed in chapter 2. On the other hand, it is more appropriate to consider the fault correction process to be related to the fault detection process as a fault can only be removed after its detection. The fault correction process can be assumed to be a delayed fault detection process. Different models have been proposed by presenting different

forms of the time delay between these two processes. Extension can be made in two directions: firstly, different NHPP models could be applied for different fault detection processes; secondly, different time-delay forms can be generated under different fault correction conditions.

3.1.1 Fault Detection Models

As reviewed in section 2 on NHPP models, software fault detection process is usually assumed to follow a non-homogeneous Poisson process, in which the intensity function is time-dependent. Given the intensity function $\lambda_d(t)$, the MVF $m_d(t)$ satisfies

$$m_d(t) = \int_0^t \lambda_d(s) ds \tag{3.1}$$

The mean value function $m_d(t)$ is the characteristic of the NHPP model. Generally, different fault detection models could be obtained by using different non-decreasing functions $m_d(t)$. There are two major classes of $m_d(t)$ used to describe different fault detection processes: concave and S-shaped models. A concave $m_d(t)$ describes the fault detection process with exponential decreasing intensity. Differently, S-shaped $m_d(t)$ describes fault detection process with increasing-then-decreasing intensity, which could be interpreted as a learning process.

The G-O model is one of the most influential NHPP software reliability models. The mean value function is given as

$$m_d(t) = a\left(1 - e^{-bt}\right), a, b > 0 \tag{3.2}$$

Where *a* is the number of faults that can be detected by the testing process, and b can be interpreted as the failure occurrence rate per fault (Goel and Okumoto, 1979).

There are many other models widely discussed, such as the Duane model, which is also referred to as the Weibull process model, and the K-stage Erlangian (gamma) growth curve model (k=3). Another widely discussed model is the delayed S-shaped model studied in Yamada et al. (1984a). It is used to model the delayed reporting phenomenon for fault detection. The mean value function is given as

$$m_d(t) = a \cdot [1 - (1 + bt)e^{-bt}],\ a, b > 0 \tag{3.3}$$

with parameter *a* denoting the number of faults to be detected and *b* corresponding to a fault detection rate.

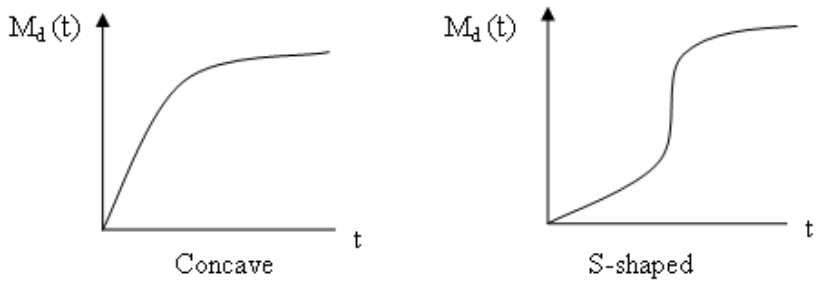


Figure 3. 1 Two classes of mean value function md(t)

3.1.2 Fault Correction Models

Fault correction is related to the fault detection process, and it can be modeled with reference to the NHPP models described previously. Specifically, a fault can be corrected only after its detection, and the fault correction process can be modeled as a delayed fault detection process. The difference between these two processes is the time delay, which is the time spent to correct the detected fault. Such delay could be deterministic or random, which in turn can also be time-dependent. Then similar to FDP models characterized with MVF of $m_d(t)$, FCP models characterized with MVF of $m_c(t)$ can be derived from $m_d(t)$ and time delay Δ.

3.1.3 Paired FDP and FCP Models

In the fault detection and correction model we proposed, we define the time delay as the correction time and denote it as Δ. This time delay can be modeled as a deterministic or random delay. In practice, it is more realistic to assume the time delay as a random variable. Similar to FDP models characterized with mean value function (MVF) $m_d(t)$,

FCP models can be characterized with MVF $m_c(t)$. The MVF of FCP models can be derived from $m_d(t)$ and the time delay Δ: $m_c(t) = \int_0^t \lambda_c(t)dt = \int_0^t E[\lambda_d(t-\Delta)]dt$. Traditional models assume perfect debugging, which means that no faults are introduced when correcting one and faults detected are immediately corrected. In our modeling, we consider the imperfect debugging by assuming that faults are not immediately corrected. There is a time lag between fault detection and fault correction process, as we need more time to locate and correct it. However, we still assume that no faults are introduced when correcting one; the total number of initial faults will not increase over time. To emphasize the fault correction modeling, the G-O model is applied to the fault detection process for illustrative purpose.

Combining the NHPP model for FDP and the correction time model related to FCP, we can get the paired FDP and FCP modeling framework based on the following assumptions.

1) The fault detection process can be described as an NHPP characterized with intensity function $\lambda_d(t)$.
2) Each detected fault will be isolated and goes into correction immediately.
3) It will take a random time for its correction.

Accordingly, the paired model will be characterized with the following paired mean value functions:

$$m_d(t) = \int_0^t \lambda_d(s)ds \tag{3.4}$$

$$m_c(t) = \int_0^t \lambda_c(t)dt = \int_0^t E[\lambda_d(t-\Delta)]dt \tag{3.5}$$

Usually, the paired model contains some unknown parameters and the estimation is carried out with the method of least squares. Specifically, against observations of fault detection and correction, the parameters are estimated by minimizing the sum of squared residuals, which is the difference between MVFs and the observations for both detected and corrected faults (summed) as

$$\sum_{i=1}^{n}\left[\left(m_d(t_i)-d_i\right)^2+\left(m_c(t_i)-c_i\right)^2\right] \tag{3.6}$$

where d_i and c_i denote the cumulative number of detected and corrected faults collected till time t_i respectively; t_i, $i = 1,2,3,\ldots,$ are the running times from the beginning of testing.

Commonly, numerical procedures have to be developed in order to obtain the LSEs (least square estimates). With the LSE of the parameters, the performance of the paired model can be evaluated through the goodness-of-fit criterion. MSE (Mean squared error) is adopted as the measurement and it is calculated through the average of MSEs for both fault detection and correction. Both MSEs are calculated through the average squared difference between the estimated expectations and actual data, as in the following equation:

$$MSE = \frac{1}{2}\left[MSE_d + MSE_c\right] = \frac{1}{2}\cdot\left[\frac{1}{n}\cdot\sum_{i=1}^{n}\left(m_d(t_i) - d_i\right)^2 + \frac{1}{n}\cdot\sum_{i=1}^{n}\left(m_c(t_i) - c_i\right)^2\right] \qquad (3.7)$$

The MSE for the combined fault detection and correction process is defined as the average value of MSE for fault detection process and MSE for fault correction process; it can be minimized with respect to the model parameters when actual data is available.

Different paired models have different parameter combinations, and traditionally, all parameters can be estimated together through least square method. In next chapter we will discuss using Maximum Likelihood Estimation (MLE) to obtain the model parameters.

3.2 Models for Fault Correction

As mentioned earlier, fault correction process is a delayed fault detection process. However, there are different types of delay models that can be used. Here we provide some discussions on different types of delay models, and we use the G-O model for FDP for the purpose of illustration. Besides the parameters from NHPP models, there are parameters for fault correction. The MLE parameter estimation method will be discussed later in the next chapter.

The deterministic assumptions on correction time are simplistic and often not realistic. In fact, software fault correction is closely related to human behavior, which is an uncertainty factor. Also, detected faults are different and their appearance sequence is

random in system testing. Therefore, it would be more practical to model the correction time with a random variable.

3.2.1 Exponentially Distributed Time Delay

The correction time approximately follows exponential distribution in many practical software testing projects (Musa et al., 1987). Assuming the correction time for each detected fault is exponentially distributed with $\Delta \sim \exp(\mu)$, then with given fault detection intensity function $\lambda_d(t)$, the fault correction density function is

$$\lambda_c(t) = E[\lambda_d(t-\Delta)] = \int_0^t \lambda_d(t-x)\cdot\mu\cdot e^{-\mu x}dx \tag{3.8}$$

The fault correction process can then be described by the following MVF:

$$m_c(t) = m_d(t-\Delta) = \int_0^t \lambda_c(t)dt \tag{3.9}$$

The fault correction MVF for GO-model is given as

$$m_c(t) = \begin{cases} a\cdot\left[1-(1+bt)e^{-bt}\right] & \mu = b \\ a\cdot\left[1-\dfrac{\mu}{\mu-b}e^{-bt}+\dfrac{b}{\mu-b}e^{-\mu t}\right] & \mu \neq b \end{cases} \tag{3.10}$$

where $m_c(t)$ has the same form as the $m_d(t)$ for S-shaped NHPP model while $\mu = b$.

3.2.2 Normally Distributed Time Delay

Assuming faults are of equal size, then we can model the time delay as a normally distributed variable with mean μ and variance σ^2, the fault correction density function given the fault detection intensity function $\lambda_d(t)$ is given as

$$\lambda_c(t) = E[\lambda_d(t-\Delta)] = \int_0^t \lambda_d(t-x)\cdot\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x-\mu)^2}{2\sigma^2}}dx = abe^{\frac{b^2+2\mu b}{2}-bt}\left[\Phi(t-\mu-b)-\Phi(-\mu-b)\right] \tag{3.11}$$

The fault correction process can then be described by the following MVF:

$$m_c(t) = m_d(t-\Delta) = \int_0^t \lambda_c(t)dt = \int_0^t abe^{\frac{b^2+2\mu b}{2}-bt}\left[\Phi(t-\mu-b)-\Phi(-\mu-b)\right]dt \tag{3.12}$$

Simplifying the above formula, we get:

$$m_c(t) = -ae^{-bt+\mu b+b^2\sigma^2/2}\left(\Phi(t,\ b\sigma^2+\mu,\ \sigma)-\Phi(0,\ b\sigma^2+\mu,\ \sigma)\right)+a\left(\Phi(t,\ \mu,\ \sigma)-\Phi(0,\ \mu,\ \sigma)\right) \tag{3.13}$$

3.2.3 Gamma Distributed Time Delay

To provide more flexible modeling of the correction processes, some extended distributions can be used for the correction time. One possible distribution is the Gamma distribution, which is the generalized form of the exponential distribution. This distribution is reasonable if the correction has to go through a few steps. In this case, the Gamma distributed time delay Δ has density

$$f(\Delta = x) = \frac{1}{\beta^{\alpha} \cdot \Gamma(\alpha)} x^{\alpha-1} e^{-x/\beta},\ \alpha,\beta > 0, x > 0 \tag{3.14}$$

Given the fault detection intensity function $\lambda_d(t)$, the fault correction density function is

$$\lambda_c(t) = E[\lambda_d(t-\Delta)] = \int_0^t \lambda_d(t-x) \cdot f(x) \cdot dx \tag{3.15}$$

Then, the fault correction mean value function for GO-model is

$$m_c(t) = \int_0^t \lambda_c(t)dt = \int_0^t \int_0^t abe^{-b(t-x)} \cdot \frac{x^{\alpha-1} e^{-x/\beta}}{\beta^{\alpha} \cdot \Gamma(\alpha)} dx \cdot dt = \frac{ab}{\beta^{\alpha} \cdot \Gamma(\alpha)} \cdot \int_0^t e^{-bt} \int_0^t \cdot x^{\alpha-1} e^{(b-1/\beta)\cdot x} \cdot dxdt \tag{3.16}$$

Simplifying the above equation we get

$$m_c(t) = a\Gamma(t,\alpha,\beta) - \frac{ae^{-bt}}{(1-b\beta)^{\alpha}} \Gamma(t,\alpha,\frac{\beta}{1-b\beta}) \tag{3.17}$$

3.3 Residual Number of Faults

With models for both fault detection and correction processes, it is more close to the reality of software testing. With inheritance of the traditional SRGMs as model for FDP, one assumption is relaxed to incorporate FCP by introducing the concept of correction time. As a whole, the software testing model is a paired model with both FDP model and FCP model separately. Such paired model can help us to study one interesting

property of the number of detected but uncorrected faults. Residual faults can be deducted as the difference between fault detection and fault correction.

3.4 Summary

In this chapter, a paired FDP and FCP modeling framework is proposed, by assuming the relationship between FDP and FCP is the time delay. Generally, modeling both fault detection and correction processes will provide more information than traditional models. Therefore, more accurate and useful analysis and decision making can be conducted. It is more realistic compared with traditional software reliability models as this proposed model takes into account of the time delay.

Chapter 4 Maximum likelihood estimation for the fault detection and correction process

Fitting a proposed model to the actual data of faults detection and correction involves estimating the model's parameters from the real test data sets. Up to now, no MLE method has been applied in the existing studies to estimate parameters in the fault detection and correction process; although it is well accepted that the MLE method is quite straightforward having many desired properties such as asymptotic normality, admissibility, robustness and consistency, and widely used to estimate the parameters for SRGMs (Xie, 1991; Zou, 2003; Inoue and Yamada, 2004). In this chapter, we consider the issue of applying the MLE method to the fault detection and correction modeling from theoretical and experimental perspectives.

4.1 Maximum Likelihood Estimation

4.1.1 Point Estimation

Usually, the paired model contains some unknown parameters and the estimation is carried out with the method of least squares.

In this section, we give a brief introduction of using maximum likelihood estimation (MLE) method to estimate parameters in the fault detection modeling. To model the fault detection process in software testing, consider a random sample $d_1, d_2, \ldots, d_T$, where d_i denotes the number of detected faults in time interval $\left[s_{i-1}, s_i\right), s_i, i \geq 0$ are the

running times from the beginning of testing, $m_d(s_i)$ is the mean value function of the fault detection model at time s_i and θ represent the parameters in this model. The cumulative number of software faults detected up to time t is assumed to be a NHPP, with independent increments, therefore, $d_i, i \geq 0$ are independent from each other. The joint density of the detected fault counts over the given partition can be obtained, and the likelihood function can be modeled:

$$L(d_1, \cdots d_k \mid \theta) = \prod_{i=1}^{k} e^{-[m_d(s_i)|_\theta - m_d(s_{i-1})|_\theta]} \frac{\left(m_d(s_i)|_\theta - m_d(s_{i-1})|_\theta\right)^{d_i}}{d_i!} \tag{4.1}$$

where the maximum likelihood (ML) estimates of model parameters are $\hat{\theta} = \arg\max_{\theta \in \Theta} \hat{L}(\theta)$.

Based on the MLE analysis of the fault detection process, we can further consider the MLE method for the fault detection and correction process. Assume by time t_i the number of detected faults is n_i and the number of corrected faults is m_i. Since the fault correction process is regarded as a delayed fault detection process, we can regard the above fault correction process as a fault detection process which by time t_i - Δ_i have detected m_i faults. The time delay $\Delta_i, i = 1,2,\ldots \overset{i.i.d.}{\sim} \Delta$, where Δ is assumed to be a random variable satisfying a certain probability distribution. It is the time delay Δ_i that decided whether the time t_i - Δ_i is before time t_{i-1} or after time t_{i-1}. The time t_i - Δ_i should be less than t_i since the total number of corrected faults can not be greater than the total number of detected faults.

We get the following characteristics of the time delay:

1) The time delay should always be non-negative. Specifically, if the time delay is equal to zero, that means the detected faults are removed immediately without further delay.
2) Regarding the correction time as a constant is too simplistic to be the case in practice. It would be more practical to model the correction time with a random variable as correction is a repair activity that will not take a fixed amount of time (Schneidewind, 2001).

In our model, we assume that any fault detected needs a correction time to be corrected. The correction process is NHPP with intensity function $\lambda_c(t) = E[\lambda_d(t-\Delta)] = \int_0^t \lambda_d(t-x)\cdot\mu\cdot e^{-\mu x}dx$ and mean value function $m_c(t) = m_d(t-\Delta) = \int_0^t \lambda_c(t)dt$. Therefore, if by time t_i we have corrected m_i faults, we can say that by time $t_i - \Delta_i$ we have detected m_i faults, if the delay time is fixed. If there are faults detected during $t_{i-1} \in (t_i - \Delta_i, t_i)$, then $t_{i-1} > t_i - \Delta_i \Leftrightarrow m_i < n_{i-1}$. This is what we have discussed in our likelihood function expression (4.2) on page 44.

Denoting n_i and m_i as the cumulative number of faults detected and corrected by time t_i respectively, and $n_0 = m_0 = 0$. Assuming the fault detection process is NHPP, we can see n_i depends on m_i. Denote $P(n_i, m_i)$ as the probability of detecting n_i faults and correcting m_i faults by time t_i. $P_d(\cdot)$ as the probability distribution function for detection

process, and $P_c(\cdot)$ as the probability distribution function for correction process. In this dependent case, conditioning on the corrected number of faults, we can rewrite the probability of detecting n_i faults and correcting m_i faults by time t_i at the very first step, denoting $\Delta_i^- = \{\Delta_i \mid \Delta_i \le t_i - t_{i-1}\}$, $\Delta_i^+ = \{\Delta_i \mid \Delta_i > t_i - t_{i-1}\}$, we have:

$$
\begin{aligned}
&P(n_i, m_i \mid n_{i-1}, m_{i-1}, \theta_0) \\
&= P(n_i \mid m_i, n_{i-1}, m_{i-1}, \theta_0) P(m_i \mid n_{i-1}, m_{i-1}, \theta_0) \\
&= \begin{Bmatrix} P_d(n_i - m_{i,[t_i-\Delta_i,\ t_i]} \mid \theta_0) P_d(m_i - n_{i-1,[t_{i-1},\ t_i-\Delta_i]} \mid \theta_0), & if\ t_{i-1} \le t_i - \Delta_i \Leftrightarrow m_i \ge n_{i-1} \\ P_d(n_i - n_{i-1,[t_{i-1},\ t_i]} \mid \theta_0) P_c(m_i - m_{i-1,[t_{i-1},\ t_i]} \mid \theta_0), & if\ t_{i-1} > t_i - \Delta_i \Leftrightarrow m_i < n_{i-1} \end{Bmatrix} \\
&= P_d(n_i - m_{i,[t_i-\Delta_i,\ t_i]} \mid \theta_0) P_d(m_i - n_{i-1,[t_{i-1},\ t_i-\Delta_i]} \mid \theta_0) I\{\Delta_i^-\} \\
&\quad + P_d(n_i - n_{i-1,[t_{i-1},\ t_i]} \mid \theta_0) P_c(m_i - m_{i-1,[t_{i-1},\ t_i]} \mid \theta_0) I\{\Delta_i^+\}
\end{aligned}
\tag{4.2}
$$

Denoting $\theta_0 \in \Theta \subset R^m$ as the parameters in the fault detection and correction modeling, then we can obtain the joint density of the detected and corrected fault counts over the given partition.

$$
\begin{aligned}
&P(n_i, m_i, i = 1, \ldots, k \mid \theta_0) = \prod_{i=1}^{k} P(n_i, m_i \mid n_{i-1}, m_{i-1}, \theta_0) \\
&= \prod_{i=1}^{k} P(n_i \mid m_i, n_{i-1}, m_{i-1}, \theta_0) P(m_i \mid n_{i-1}, m_{i-1}, \theta_0) \\
&= \prod_{i=1}^{k} \begin{Bmatrix} \{P_d(n_i - m_{i,[t_i-\Delta_i,\ t_i]} \mid \theta_0) P_d(m_i - n_{i-1,[t_{i-1},\ t_i-\Delta_i]} \mid \theta_0)\} I(\Delta_i^-) \\ + P_d(n_i - n_{i-1,[t_{i-1},\ t_i]} \mid \theta_0) P_c(m_i - m_{i-1,[t_{i-1},\ t_i]} \mid \theta_0) I(\Delta_i^+) \end{Bmatrix} \\
&= \prod_{\substack{i=1,\ldots,k \\ m_i > n_{i-1}}} P_d(n_i - m_{i,[t_i-\Delta_i,\ t_i]} \mid \theta_0) P_d(m_i - n_{i-1,[t_{i-1},\ t_i-\Delta_i]} \mid \theta_0) \\
&\qquad \times \prod_{\substack{i=1,\ldots,k \\ m_i < n_{i-1}}} P_d(n_i - n_{i-1,[t_{i-1},\ t_i]} \mid \theta_0) P_c(m_i - m_{i-1,[t_{i-1},\ t_i]} \mid \theta_0)
\end{aligned}
\tag{4.3}
$$

The likelihood function in this case is defined as this joint density, with θ_0 replaced by θ, simplifying the above equation we get:

$$
\begin{aligned}
L &= f\left(n_i, m_i, i = 1, \ldots, k \mid \theta\right) \\
&= \prod_{\substack{i=1,\ldots,k, \\ m_i > n_{i-1}}} e^{-\left[m_d(t_i) - m_d(t_{i-1}) \mid \theta\right]} \frac{\left[m_d(t_i) - m_c(t_i) \mid \theta\right]^{n_i - m_i}}{(n_i - m_i)!} \frac{\left[m_c(t_i) - m_d(t_{i-1}) \mid \theta\right]^{m_i - n_{i-1}}}{(m_i - n_{i-1})!} \\
&\times \prod_{\substack{i=1,\ldots,k, \\ m_i < n_{i-1}}} e^{-\left[m_d(t_i) - m_d(t_{i-1}) \mid \theta\right]} \frac{\left[m_d(t_i) - m_d(t_{i-1}) \mid \theta\right]^{n_i - n_{i-1}}}{(n_i - n_{i-1})!} \cdot e^{-\left[m_c(t_i) - m_c(t_{i-1}) \mid \theta\right]} \frac{\left[m_c(t_i) - m_c(t_{i-1}) \mid \theta\right]^{m_i - m_{i-1}}}{(m_i - m_{i-1})!}
\end{aligned}
$$

(4.4)

A general form of the likelihood function for the combined fault detection and correction process is given in Eq. (4.4). Under various time delay assumption, the maximum likelihood (ML) estimates of θ_0 can be obtained as $\hat{\theta} = \arg\max_{\theta \in \Theta} \hat{L}(\theta)$.

Once the estimates of all the parameters are obtained, we can use the invariance property of the MLEs to estimate other reliability measures by replacing the respective parameters according to their corresponding ML estimates. An example is the estimation of the failure intensity function. Jeske and Pham (2001) showed that the failure rate of the software at time *T* was a function of the fundamental parameters of the G-O model, and its ML estimate was consistent.

4.1.2 Interval Estimation

Denote L as the likelihood function in Eq. (4.4), and denote $\theta = (a, b, \mu)^T$. To obtain approximate confidence limits for model parameters $\theta = (a, b, \mu)^T$, the Fisher Information

matrix can be calculated to obtain the asymptotic variances and covariance of the ML estimates of the parameters.

The Fisher information matrix for the three parameters of the fault detection and correction process is

$$F = \begin{bmatrix} -E\left[\frac{\partial^2 \ln L}{\partial a^2}\right] & -E\left[\frac{\partial^2 \ln L}{\partial a \partial b}\right] & -E\left[\frac{\partial^2 \ln L}{\partial a \partial \mu}\right] \\ -E\left[\frac{\partial^2 \ln L}{\partial a \partial b}\right] & -E\left[\frac{\partial^2 \ln L}{\partial b^2}\right] & -E\left[\frac{\partial^2 \ln L}{\partial b \partial \mu}\right] \\ -E\left[\frac{\partial^2 \ln L}{\partial a \partial \mu}\right] & -E\left[\frac{\partial^2 \ln L}{\partial b \partial \mu}\right] & -E\left[\frac{\partial^2 \ln L}{\partial \mu^2}\right] \end{bmatrix} \tag{4.5}$$

The asymptotic covariance matrix V of the ML estimates for parameters $\theta = (a, b, \mu)^T$ is the inverse of the Fisher information matrix

$$V = F^{-1} = \begin{bmatrix} Var(\hat{a}) & Cov(\hat{a}, \hat{b}) & Cov(\hat{a}, \hat{\mu}) \\ Cov(\hat{a}, \hat{b}) & Var(\hat{b}) & Cov(\hat{b}, \hat{\mu}) \\ Cov(\hat{a}, \hat{\mu}) & Cov(\hat{b}, \hat{\mu}) & Var(\hat{\mu}) \end{bmatrix} \tag{4.6}$$

Employing large-sample *s*-normal distribution approximations, the two sided approximate 100 α % confidence limits for model parameters $\omega = \{a, b, \mu\}$ can be obtained as

$$\begin{aligned} \omega_{upper} &= \hat{\omega} + Z_\alpha \sqrt{Var(\hat{\omega})} \\ \omega_{lower} &= \hat{\omega} - Z_\alpha \sqrt{Var(\hat{\omega})} \end{aligned} \tag{4.7}$$

where Z_α is the (1- α) quantile of the standard *s*-normal distribution.; the model parameters ω can be $a, b,$ or μ .

Notice that the LS estimation method implicitly assumes normally distributed error. It usually has no basis for constructing confidence intervals or testing hypothesis; whereas both are naturally built into the ML estimation method.

4.1.3 Modified Likelihood Function Based On Execution Time

The likelihood function in Eq. (4.4) can be slightly modified and applied to other time units such as execution time. Similarly, if by the cumulative computer execution time t_i, we have corrected m_i faults, we can say that by time $t_i - \Delta_i$ we have detected m_i faults, if the delay time is fixed. The time $t_i - \Delta_i$ will occur during two sequential computer execution time $[t_{j-1}, t_j]$.

The probability that by time t_i - Δ_i, m_i faults are detected can be expressed as the product of two parts as in

$$P = P_d\left(n_j - m_i, [t_i - \Delta_i,\ t_j] \mid \theta_0\right) P_d\left(m_i - n_{j-1}, [t_{j-1},\ t_i - \Delta_i] \mid \theta_0\right) \tag{4.8}$$

Denoting $\Delta_i^- = \{\Delta_i \mid \Delta_i \le t_i - t_{i-1}\}$, $\Delta_i^+ = \{\Delta_i \mid \Delta_i > t_i - t_{i-1}\}$, $\theta_0 \in \Theta \subset R^m$ as the parameters in the fault detection and correction modeling, with θ_0 replaced by θ, the

likelihood function of the detected and corrected fault counts over the given partition can be obtained.

$$
\begin{aligned}
L &= f\left(n_i, m_i, i=1,\ldots,k \mid \theta\right) \\
&= \prod_{\substack{i=1,\cdots k, \\ m_i > n_{i-1}}} e^{-\left[m_d(t_i)-m_d(t_{i-1})\mid\theta\right]} \frac{\left[m_d(t_i)-m_c(t_i)\mid\theta\right]^{n_i-m_i}}{(n_i-m_i)!} \frac{\left[m_c(t_i)-m_d(t_{i-1})\mid\theta\right]^{m_i-n_{i-1}}}{(m_i-n_{i-1})!} \\
&\times \prod_{\substack{i=1,\cdots k, \\ m_i < n_{i-1}}} e^{-\left[m_d(t_i)-m_d(t_{i-1})\mid\theta\right]} \frac{\left[m_d(t_i)-m_d(t_{i-1})\mid\theta\right]^{n_i-n_{i-1}}}{(n_i-n_{i-1})!} \cdot e^{-\left[m_d(t_i)-m_c(t_{j-1})\mid\theta\right]} \\
&\frac{\left[m_d(t_j)-m_c(t_i)\mid\theta\right]^{n_j-m_i}}{(n_j-m_i)!} \frac{\left[m_c(t_i)-m_d(t_{j-1})\mid\theta\right]^{m_i-n_{j-1}}}{(m_i-n_{j-1})!}
\end{aligned}
\tag{4.9}
$$

Under various time delay assumption, the maximum likelihood estimates of θ_0 can be obtained as $\hat{\theta} = \arg\max_{\theta\in\Theta} \hat{L}(\theta)$.

4.2 Numerical Application

Below we use the proposed fault detection and correction models to model a real data set obtained from the testing process of a medium-sized software project, and then apply the proposed MLE approach to estimate the parameters within the model. Different from traditional software reliability data set, this dataset includes not only fault detection data but also fault correction data. However, there is no tag information that indicates when a certain fault is corrected, and only grouped data on the number of faults per week is available. Usually, software reliability models are applied at the late phase of testing, and related analysis will be updated with newly collected data.

Our analysis is based on the current stage of software testing, with all 17 available data points. The data set in Table 4.1 is from the testing process on a medium-sized software project and it counts the number of faults per week (Xie et al., 2007). Different from traditional dataset, this includes both fault detection data and fault correction data. In Table 4.1, $\Delta d(t)$ denotes the incremental detected defaults per week; $d(t)$ denotes cumulative detected defaults by time t; $\Delta c(t)$ denotes the incremental corrected defaults per week; $c(t)$ denotes cumulative corrected defaults by time t.

Table 4. 1 Fault detection and correction data (incremental and cumulative faults)

Week t	$\Delta d(t)$	$d(t)$	$\Delta c(t)$	$c(t)$
1	12	12	3	3
2	11	23	0	3
3	20	43	9	12
4	21	64	20	32
5	20	84	21	53
6	13	97	25	78
7	12	109	11	89
8	2	111	9	98
9	1	112	9	107
10	2	114	2	109
11	2	116	4	113
12	7	123	7	120
13	3	126	5	125
14	2	128	2	127
15	4	132	0	127
16	9	141	8	135
17	3	144	8	143

To highlight the idea and approach, we apply the G-O model as an example, although any other software reliability growth model (Xie, 1991) can be used. To illustrate the application, GO-model paired with various fault correction models (due to various time delay distribution) are tried with the dataset in Table 4.1. The proposed approach of ML

estimation is carried out. The information matrix can then be used to obtain an estimate of the parameter variance.

4.2.1 ML Estimation

Exponential time delay. Assuming an exponentially distributed correction time for each detected fault with $\Delta \sim \exp(\mu)$, the ML estimates for a, b, and μ are $\hat{a}$=165, $\hat{b}$=0.12, and $\hat{\mu}$=1.63, respectively. Table 4.2 shows the data sets of detected and corrected number of faults from the software testing process of a middle-sized software project. The fitted values compared with the actual data are given in Table 4.2. In addition to the actual observed faults number (the actual detected faults No. and the actual corrected faults No.), using our proposed model we can estimate the number of faults detected and corrected correspondingly.

Table 4. 2 The fitted dataset with exponential time delay

week	actual detected fault No.	estimated detected fault No.	actual corrected fault No.	estimated corrected fault No.
1	12	18.66	3	9.60
2	23	35.21	3	25.40
3	43	49.88	12	40.83
4	64	62.90	32	54.81
5	84	74.45	53	67.25
6	97	84.69	78	78.30
7	109	93.77	89	88.11
8	111	101.82	98	96.80
9	112	108.97	107	104.51
10	114	115.30	109	111.35
11	116	120.92	113	117.42
12	123	125.91	120	122.80
13	126	130.33	125	127.57
14	128	134.25	127	131.80
15	132	137.73	127	135.56
16	141	140.81	135	138.89
17	144	143.55	143	141.84

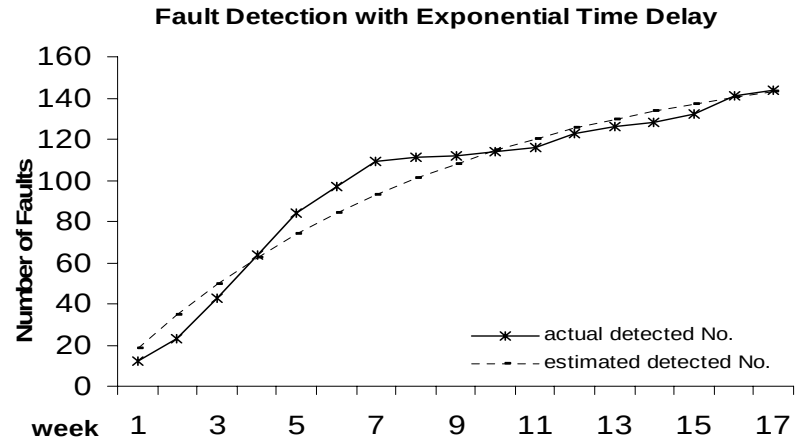
As the purpose is to highlight our idea and approach of the fault detection and correction modeling, we are proposing our model based on G-O model; while in reality, it might be better to use S-shaped model or other more complex models which can fit the real situation better.

Using Eq. (4.4) as an approximation only, the asymptotic covariance matrix of the ML estimates $\theta = (a, b, \mu)^T$ is calculated, and given as

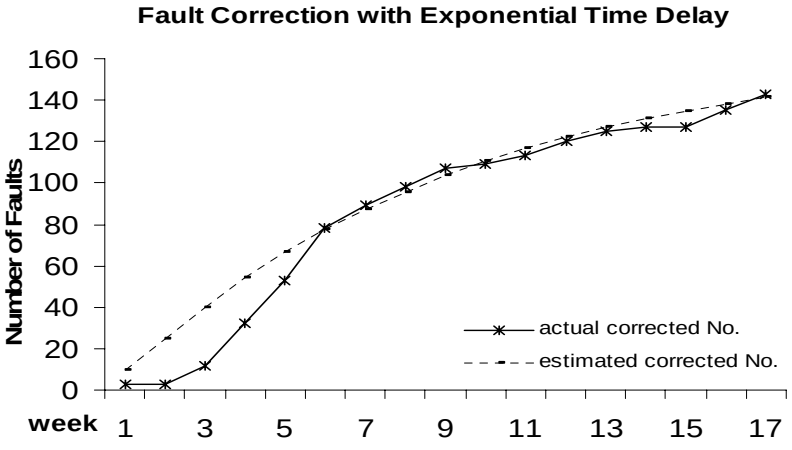
$$Var(\hat{\theta}) = I^{-1}(\hat{\theta}) = \begin{bmatrix} 355.80 & -0.32 & -0.12 \\ -0.32 & 0.0004 & 0.0002 \\ -0.12 & 0.0002 & 0.04 \end{bmatrix} \tag{4.10}$$

In this case, the ML estimator of a is 165, where the parameter a is related to the number of total faults in the software. The 95% confidence interval is [128, 202]. Notice that this confidence interval sees very wide. Nayak et al. (2008) discussed certain parameter-based asymptotic properties of the ML estimators of the model parameters and some logical implications of NHPP model assumptions; while there are also some limitations of the ML estimates. As discussed by Jeske and Pham (2001), the failure rate of the software at time T was a function of the fundamental parameters of the G-O model, and its ML estimate was consistent; however, the ML estimate of parameter a of the G-O model was not consistent when the observation period extends to infinity. The reason could be that in reality, the testing time can never be infinity. This could be one of our future topics.

The parameter b is interpreted as the testing efficiency, and it is related to the reliability growth rate in the testing. The ML estimator for b is 0.12, and the 95% confidence interval is [0.08, 0.16]. The parameter μ is interpreted as the expected mean value of the exponentially distributed time delay. For parameter μ, the ML estimate is 1.63, and the 95% confidence interval is [1.23, 1.96]. The goodness-of-fits for FDP and FCP are shown graphically in Figure 4.1.



(1)



(2)

Figure 4. 1 Actual versus fitted number of faults with exponential time delay

From Figure 4.1 we notice that except for the first few data points, the model can fit very well. As discussed earlier, to highlight our idea and approach, we start with the simple case by using G-O model for illustration. In reality, it might be better to use S-shaped model or other more complex models which can take into more factors and fit the data better. However, here we are using G-O model to illustrate our approach so as to have a clear understanding of the method, and results seem to be satisfactory. In our future research, we can apply our method based on more complex models and taking into consideration many other factors as well, such as testing-effort, change-point. Furthermore, the prediction performance of the model is more important compared with goodness-of-fit, which in chapter 5 we will have some further discussion.

Once the model parameters are estimated, the confidence intervals for FDP and FCP can be derived. In Figure 4.2, the confidence intervals using MLE and LSE techniques are shown.

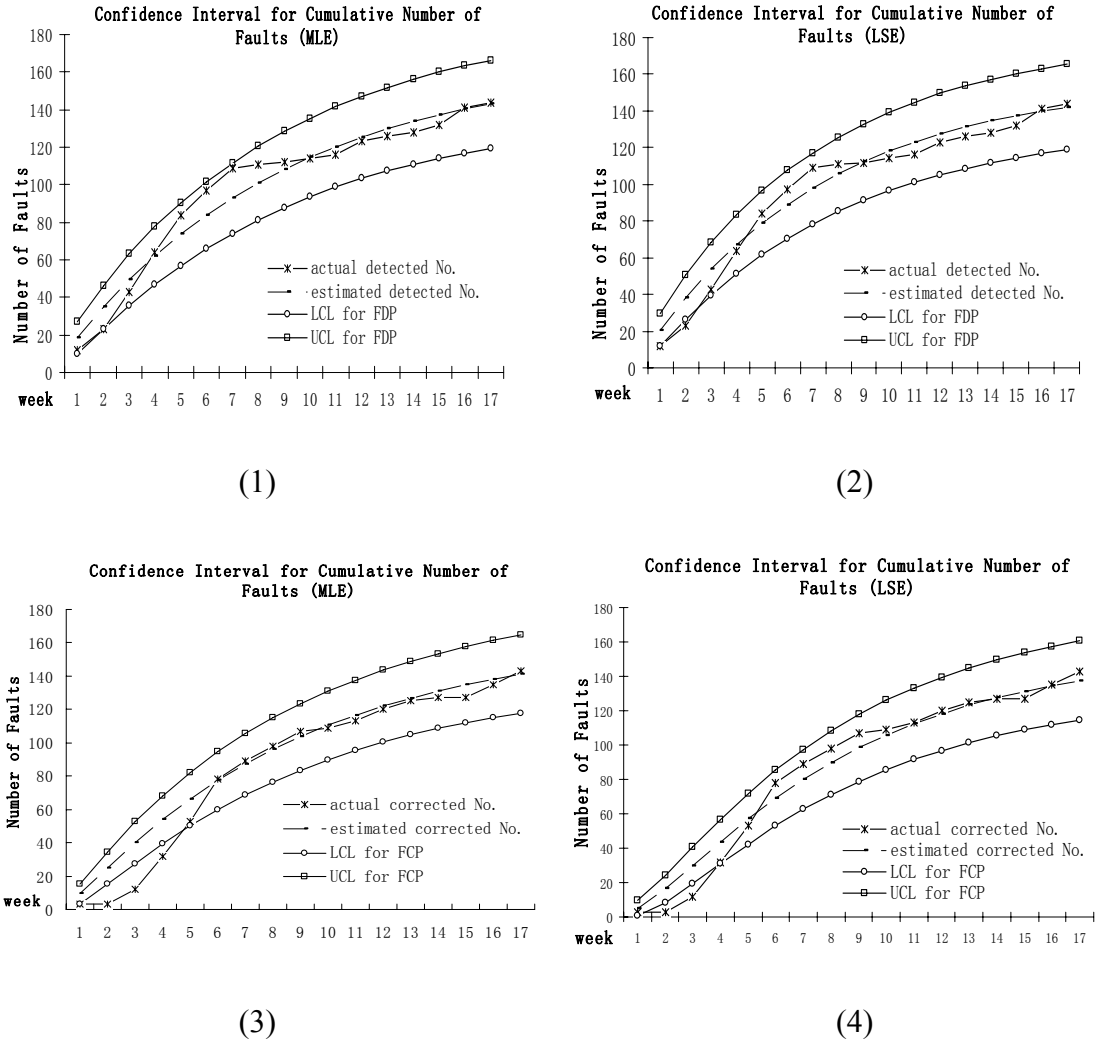
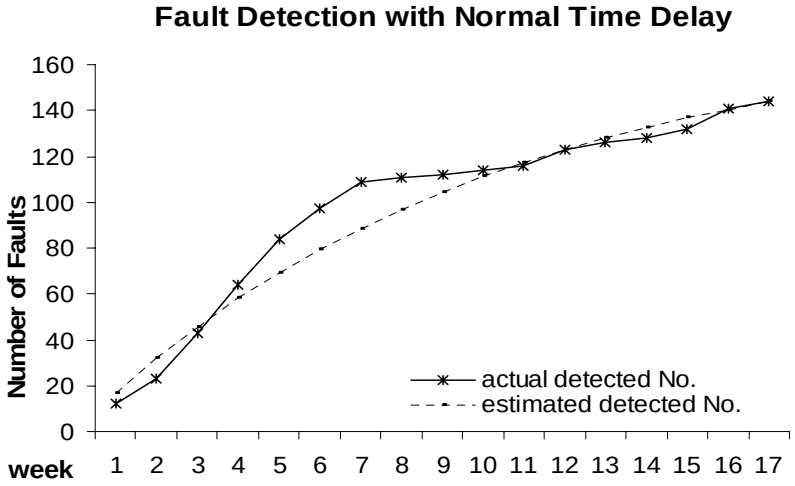


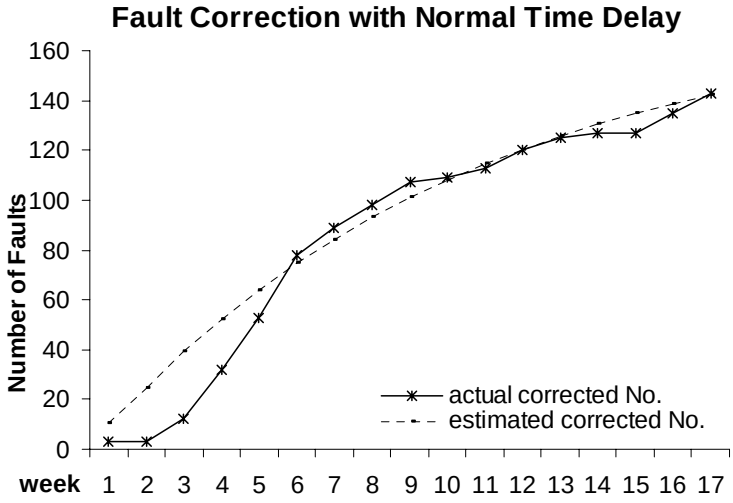
Figure 4. 2 Confidence Interval based on MLE and LSE with exponential time delay

In Figure 4.2, confidence intervals are derived based on MLE and LSE techniques separately, for both fault detection process and fault correction process. From a goodness-of-fit test point of view, the confidence intervals derived from LSE technique are comparable to MLE. For fault detection process MLE is better; while for fault correction process LSE is better.

***S-n*ormally distributed time delay.** Following the same procedure, assuming the faults are of equal size (Xie and Zhao, 1992), that is, each fault contributes the same amount to the software failure probability, the time delay can be modeled as a *s*-normally distributed variable with mean μ ,and variance σ^2 . The plots are shown in Figure 4.3.



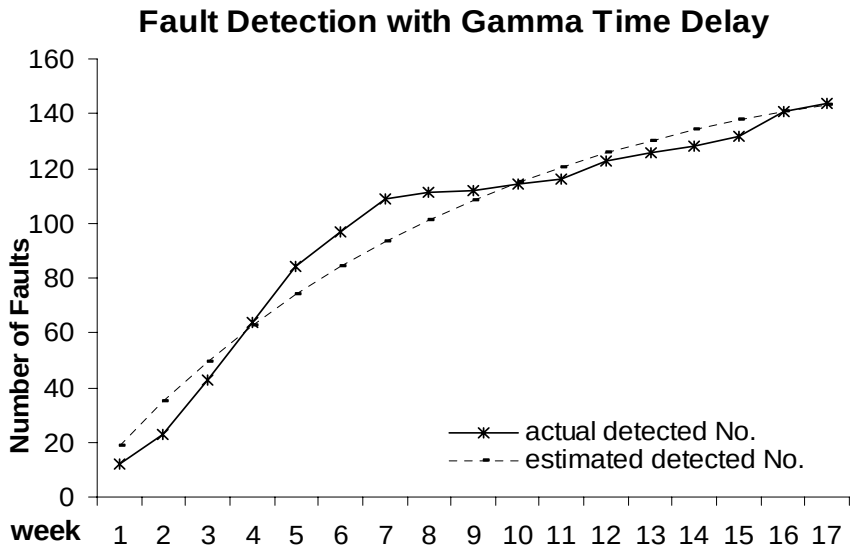
(1)



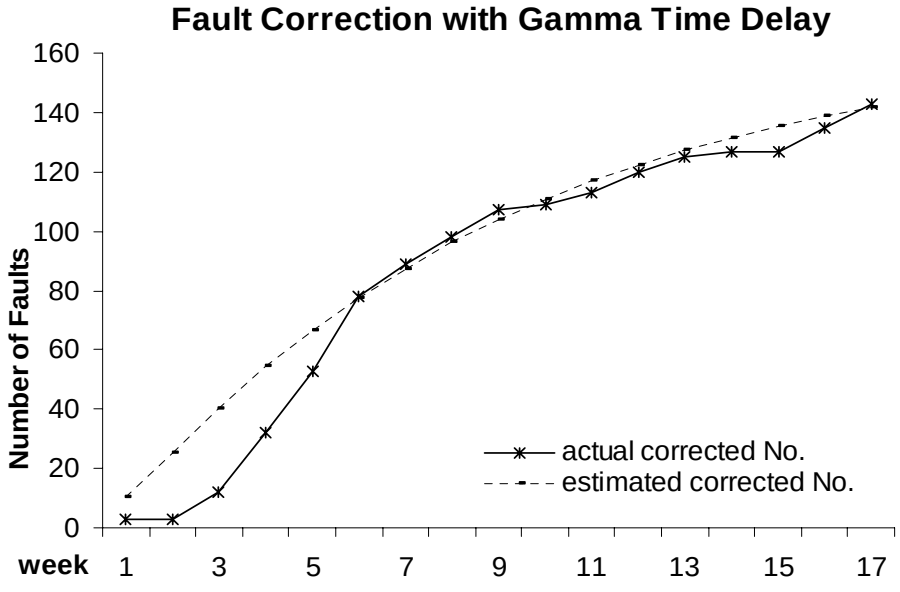
(2)

Figure 4. 3 Actual versus fitted number of faults with *s*-normally distributed time delay

Gamma time delay. Assuming the random time delay to be a Gamma distributed random variable $\Delta \sim \Gamma(\alpha, \beta)$, the plots are shown in Figure 4.4.



(1)



(2)

Figure 4. 4 Actual versus fitted number of faults with Gamma time delay

From the above analysis, three paired models are applied to fit against the real data by assuming different time delay distributions. Overall, the results showed a good agreement between actual datasets and estimated datasets under the assumption of time delay. In addition, the results of a good fitness to the dataset at the later stage of the software testing imply better predictions, and probably mean better decision making processes to a software manager (Huang and Lin, 2006; Lyu, 1996; Pham, 2000; Huang and Lyu, 2005a). The results of the estimation and the corresponding goodness-of-fit for all models are shown in Table 4.3. We use MSE_d to denote the mean squares of errors of fault detection process, MSE_c to denote the mean squares of errors of fault correction process, MSE to denote the mean squares of errors of the combined fault detection and correction process.

Table 4. 3 Summary of paired model estimates, and goodness-of-fit

Model	Estimates	MSE
Pair 1: Exponential time delay	$\hat{a}$ =165 $\hat{b}$ =0.12 $\hat{\mu}$ =1.63	MSE_d=55.02 MSE_c=132.98 MSE=94
Pair 2: Normally distributed time delay	$\hat{a}$ =176 $\hat{b}$ =0.1 $\hat{\mu}$ =0.53, $\hat{\sigma}$ =0.1	MSE_d=82.38 MSE_c=103.84 MSE=93.11
Pair 3: Gamma time delay	$\hat{a}$ =166 $\hat{b}$ =0.12 $\hat{\alpha}$ =0.73, $\hat{\beta}$ =0.83	MSE_d=55.14 MSE_c=140.64 MSE=97.89

As can be seen from Table 4.3, the pair 1 model provides the best fit for this FDP data set, and the pair 2 model provides the best fit for this FCP data set. The pair 2 model, composed of the GO-FDP model and the FCP model, provides the best fit for this whole

data set. However, it should be noted that only one dataset is used here, and the purpose is to illustrate the procedure of application.

Figure 4.5 below shows that the estimated fault correction data using the ML estimation method under various time-delay forms can fit the actual fault correction data well after the 5th week.

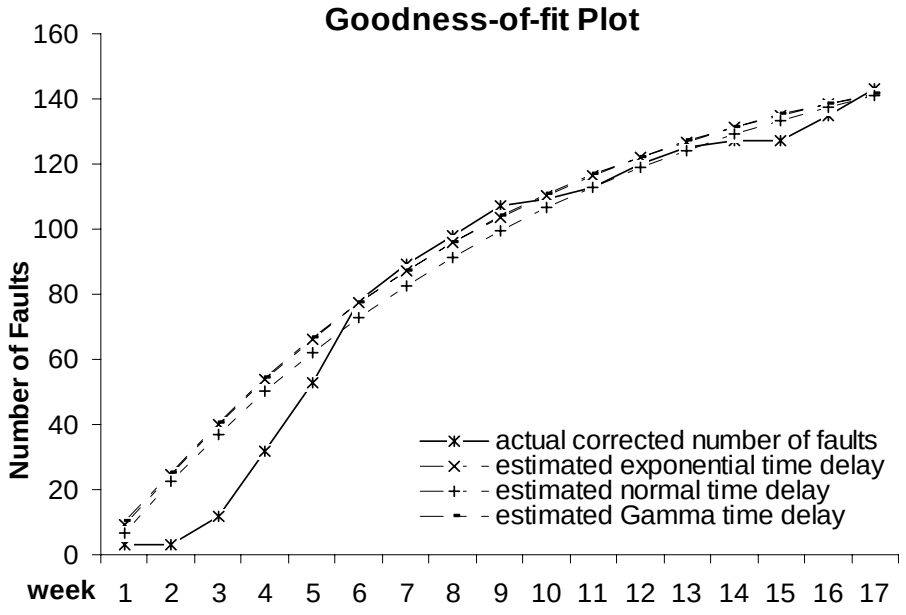


Figure 4. 5 Plot of the goodness-of-fit for the FCP under various time-delay forms

4.2.2 ML Estimates Based On Modified Likelihood Function

Instead of using the likelihood function Eq. (4.4) to obtain the ML estimates, the revised likelihood function Eq. (4.9) is used to obtain the ML estimates for the FDP and FCP modeling analysis.

Exponential time delay. Assuming the correction time for each detected fault is exponentially distributed with $\Delta \sim \exp(\mu)$, the ML estimates for a, b, and μ are $\hat{a}$=158, $\hat{b}$=0.14, and $\hat{\mu}$=0.64, respectively. The results are shown in Figure 4.6.

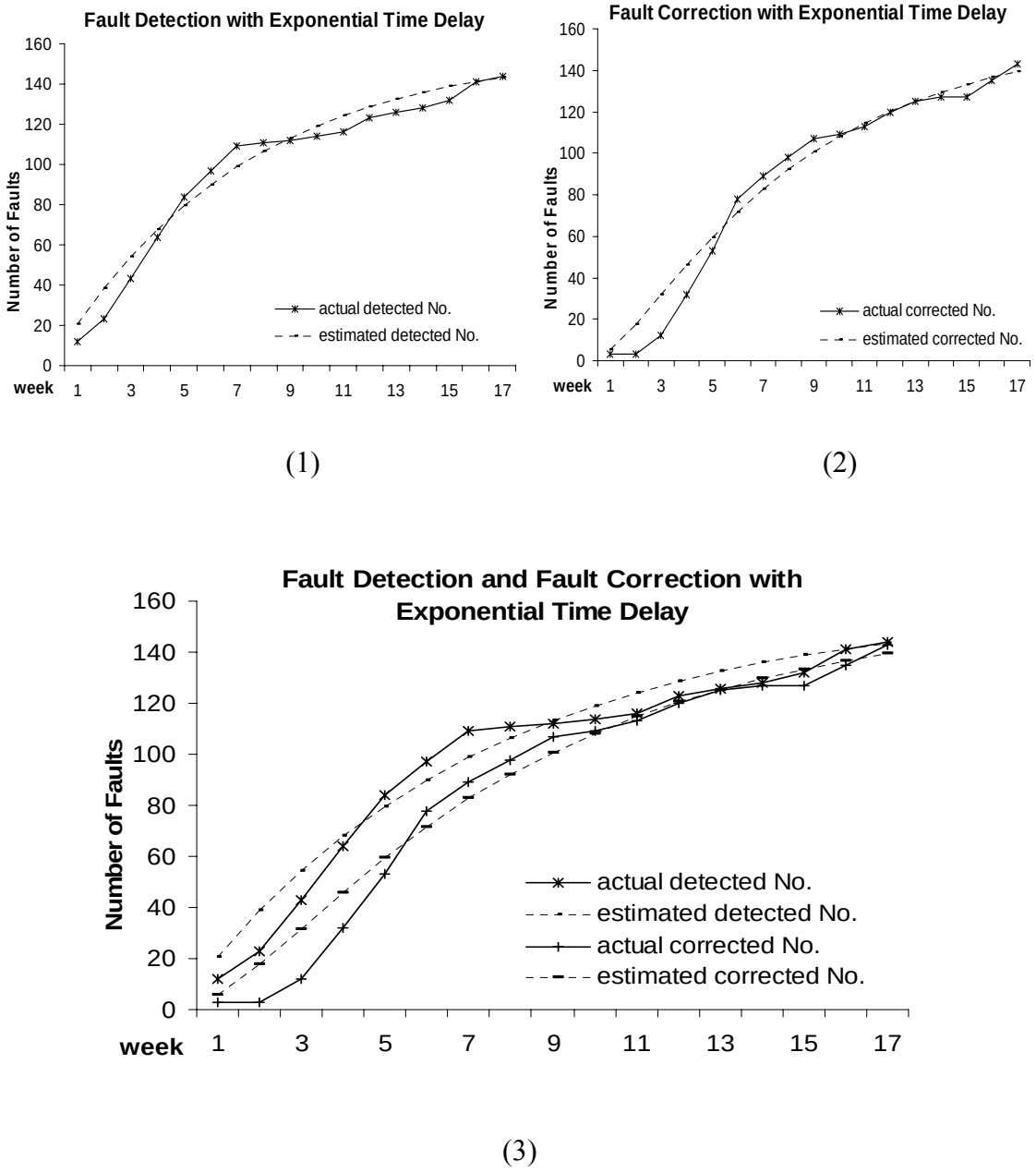


Figure 4. 6 Actual versus fitted number of faults with exponential time delay with revised likelihood function

***S*-normally distributed time delay.** Assuming the correction time for each detected fault is *s*-normally distributed with $\Delta \sim N(\mu, \sigma^2)$, the ML estimates for a, b, μ, and σ are $\hat{a}$=164, $\hat{b}$=0.14, $\hat{\mu}$=2.75, and $\hat{\sigma}$=1.04, respectively. The results are shown in Figure 4.7.

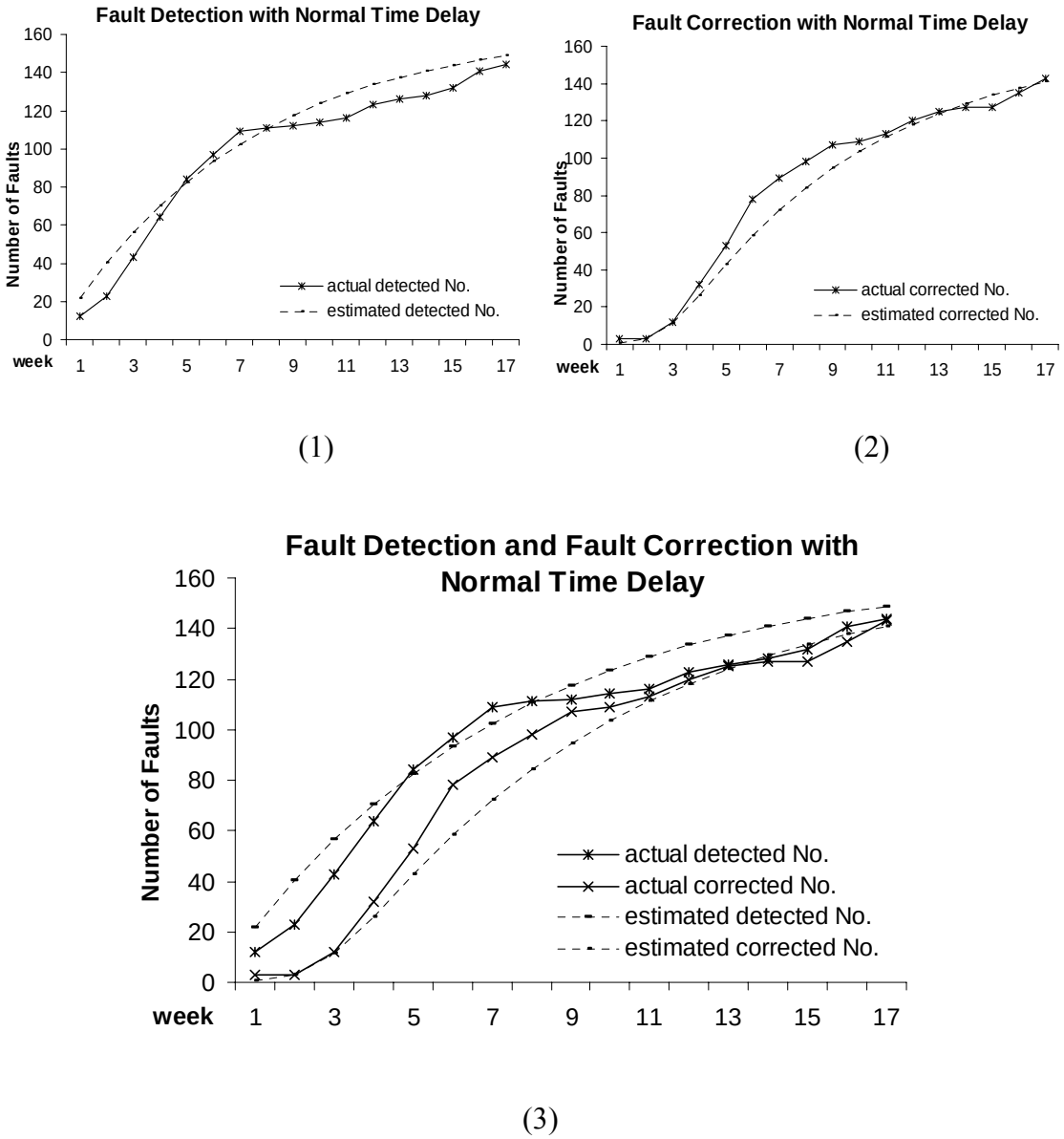


Figure 4. 7 Actual versus fitted number of faults with *s*-normally distributed time delay with revised likelihood function

Gamma time delay. Assume the random time delay to be Gamma distributed $\Delta \sim \Gamma(\alpha, \beta)$. Then the ML estimates of a, b, α, and β are $\hat{a}$=165, $\hat{b}$=0.12, $\hat{\alpha}$=1.12, and $\hat{\beta}$=0.56, respectively. The results are shown in Figure 4.8.

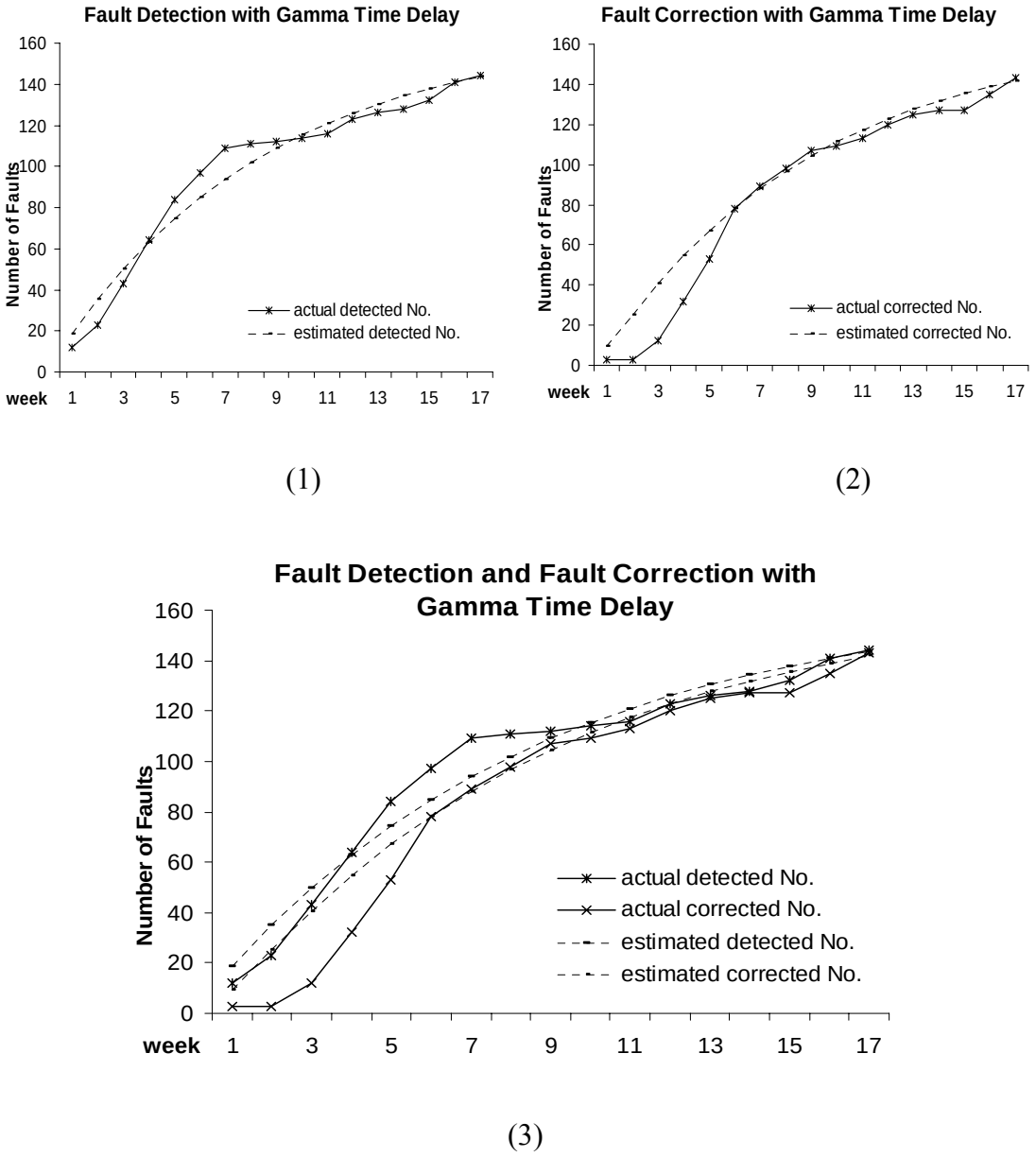


Figure 4. 8 Actual versus fitted number of faults with Gamma time delay with revised likelihood function

4.3 Summary

From above analysis, three paired models by considering different time delay function are applied to fit against the real data using revised likelihood function Eq. (4.9) instead of Eq. (4.4). The results of the estimation, and the corresponding goodness-of-fit comparing likelihood function Eq. (4.4) with Eq. (4.9), are listed in Table 4.4.

The revised likelihood function is more complex and thus more accurate; therefore we are expecting that the likelihood function with Eq. (4.9) should provide less variance and be a better choice. However, in practice, software manager may prefer the one with Eq. (4.4) instead, as it may not be that accurate but it is much easier to calculate and apply.

Table 4. 4 Comparison of paired model estimates, and goodness-of-fit

Model	ML estimates of Eq. (4.4)	MSE of Eq. (4.4)	ML estimates of Eq. (4.9)	MSE of Eq. (4.9)
Pair 1: Exponential time delay	$\hat{a}$=165 $\hat{b}$=0.12 $\hat{\mu}$=1.63	MSE=94	$\hat{a}$=158 $\hat{b}$=0.14 $\hat{\mu}$=10.64	MSE=58.11
Pair 2: Normally distributed time delay	$\hat{a}$ =176 $\hat{b}$=0.1 $\hat{\mu}$=0.53, $\hat{\sigma}$=0.1	MSE=93.11	$\hat{a}$ =164 $\hat{b}$=0.14 $\hat{\mu}$=2.75, $\hat{\sigma}$=1.04	MSE=82.54
Pair 3: Gamma time delay	$\hat{a}$ =166 $\hat{b}$=0.12 $\hat{\alpha}$=0.73, $\hat{\beta}$=0.83	MSE=97.89	$\hat{a}$ =165 $\hat{b}$=0.12 $\hat{\alpha}$=1.12, $\hat{\beta}$=0.56	MSE=92.78

As can be seen from Table 4.4, based on revised likelihood function Eq. (4.9), the pair 1 model, composed of the GO-FDP model and the FCP model, provides the best fit for the whole dataset; while for the likelihood function Eq. (4.4), the best fit is the pair 2 model. For this dataset the revised likelihood function based on execution time can give a better goodness-of-fit compared with the one based on calendar time.

This result is within our expectation. Execution time and calendar time have been an important topic in many papers (Musa and Okumoto, 1987; Xie, 1991). Execution time is useful to obtain more accurate estimation compared with calendar time, as software testing is in a sense much more uniform in execution time than in calendar time. However, from the management point of view, calendar time models are more efficient for the allocation of testing resource and for the determination of optimal release time. It is possible to combine calendar time and execution time for many different software reliability models and it could be one interesting topic for future study.

Chapter 5 Prediction Analysis of FDP FCP model

As pointed out by Li et al. (2007), building good reliability models is one of the key problems in the field of software reliability. A good software reliability model should give good predictions of future failure behavior, compute useful quantities and be widely applicable. Therefore, a very important goal of current software reliability research is to develop general prediction models (Karunanithi et al., 1992). For a software manager, it is essential to be able to predict the future behavior of the fault detection and correction process. The prediction is important for the allocation of further testing resources and for the study of software release problems. No single measure alone is adequate to determine the best parameter estimation method on a given dataset. In fact, software managers would prefer to see gradually smaller percentage of the data predicted, because the further the testing process running, the more expensive it becomes. Therefore, our main attention is to give good predictions to the later stage of software testing.

5.1 Prediction Performance

To study the predictive capability, based on our proposed FDP and FCP model, the ML estimates are obtained based on the proposed likelihood function Eq. (4.4) for illustration purposes only. Assuming exponential time delay, predictions for 5 weeks ahead is showed here, i.e., using the data of the first 12 weeks to predict the rest 5 weeks. The predicted value is then compared with the real observed dataset.

Defining the predictive validity as $RE = \frac{m(t_i) - N_i}{N_i}$, where N_i is the faults observed by time t_i, the comparison criterion used here is the mean of relative errors (MRE) defined as

$$MRE = \frac{1}{k}\sum_{i=1}^{k}\left|\frac{m(t_i) - N_i}{N_i}\right| \tag{5.1}$$

This measure shows the capability of the model to predict the fault behavior (Musa et al., 1987), and a lower value of MRE indicates better predictive performance. We calculate the MRE for fault detection process, and fault correction process separately, then we use the weighted MRE (the average of the two MRE) as the criterion to judge the whole fault detection and correction processes.

Based on the first 12 weeks data from (Xie et al., 2007), the ML estimates based on likelihood function Eq. (4.4) are $\hat{a}$=163, $\hat{b}$=0.12, and $\hat{\mu}$=1.37. The results of goodness-of-fit of the first 12 weeks data are given in Table 5.1. The predictions for the next 5 weeks based on this model are then compared with the actual observations as shown in Table 5.1, and plotted in Figure 5.1. For the data of the remaining 5 weeks, the MRE for the detection process is 0.026, and the MRE for the correction process 0.027. The average MRE is 0.026.

Table 5. 1 Goodness-of-fit and prediction using first 12 dataset with MLE

	Goodness-of-fit (MLE)			
Week	Actual detected No.	Estimated detected No.	Actual corrected No.	Estimated corrected No.
1	12	18.80	3	8.69
2	23	35.43	3	23.91
3	43	50.13	12	39.29
4	64	63.14	32	53.38
5	84	74.64	53	65.97
6	97	84.80	78	77.13
7	109	93.79	89	87.00
8	111	101.75	98	95.74
9	112	108.78	107	103.47
10	114	114.99	109	110.30
11	116	120.49	113	116.34
12	123	125.35	120	121.68
	Prediction using first 12 data points (MLE)			
13	126	129.65	125	126.41
14	128	133.46	127	130.58
15	132	136.82	127	134.28
16	141	139.79	135	137.54
17	144	142.42	143	140.43

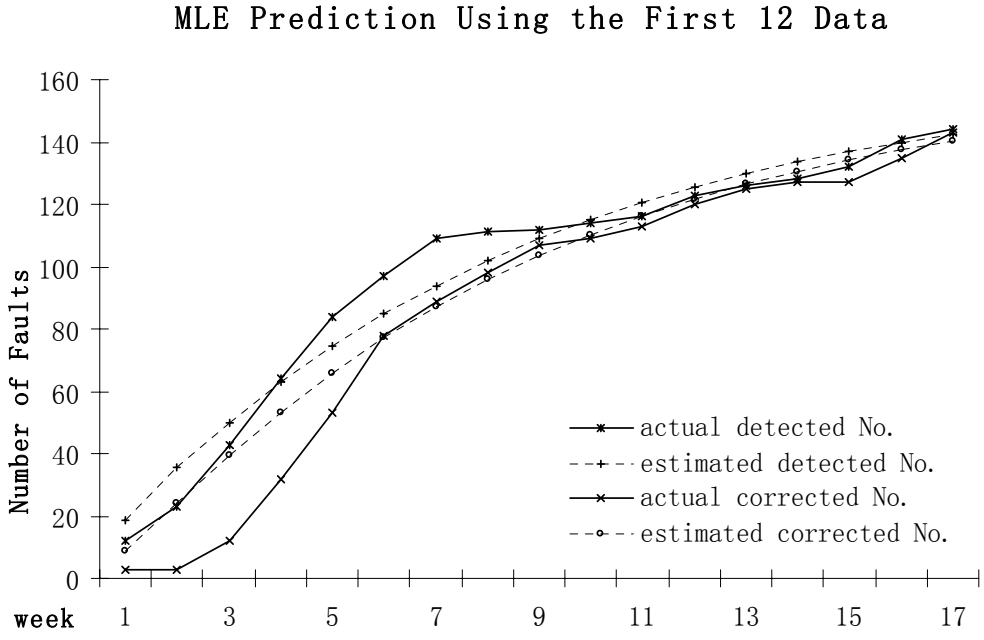


Figure 5. 1 ML estimators prediction using data of the first 12 weeks

To have a general idea of the predictive performance of the ML estimates, the LS estimates are studied in a similar way, and then compared with the ML estimates. Using the same data set of the first 12 weeks, the LS estimates are obtained as $\hat{a}$=165, $\hat{b}$=0.13, and $\hat{\mu}$=0.58. The results of goodness-of-fit of the first 12 weeks data are given in Table 6. Predictions of the last 5 weeks are compared with the actual observations as shown in Table 5.2, and plotted in Figure 5.2. The MRE computed for the 5 predicted points for the detection process is 0.053, while the MRE for the correction process is 0.025. The average MRE is 0.039.

Table 5. 2 Goodness-of-fit and prediction using first 12 data points with LSE

	Goodness-of-fit (LSE)			
Week t	Actual detected No.	Estimated detected No.	Actual corrected No.	Estimated corrected No.
1	12	19.97	3	4.90
2	23	37.54	3	15.82
3	43	52.98	12	29.14
4	64	66.56	32	42.93
5	84	78.51	53	56.23
6	97	89.01	78	68.58
7	109	98.24	89	79.81
8	111	106.36	98	89.89
9	112	113.50	107	98.87
10	114	119.78	109	106.83
11	116	125.30	113	113.87
12	123	130.16	120	120.08
	Prediction using first 12 data points (LSE)			
13	126	134.43	125	125.55
14	128	138.18	127	130.37
15	132	141.48	127	134.60
16	141	144.38	135	138.33
17	144	146.94	143	141.62

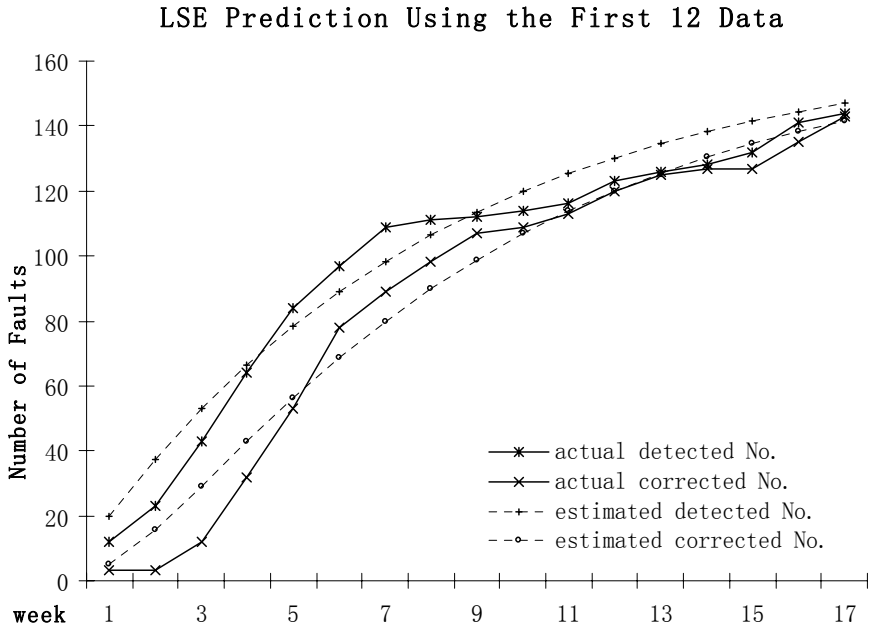
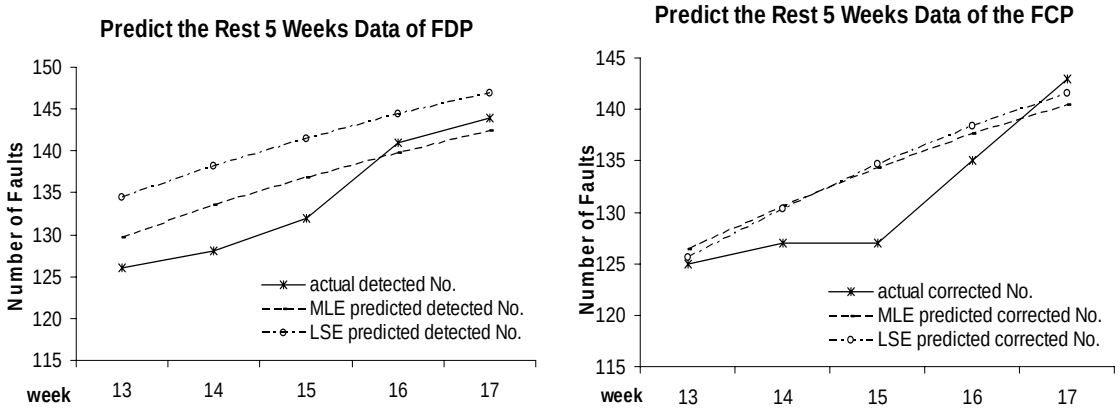


Figure 5. 2 LS estimation prediction using data of the first 12 weeks



(1) (2)

Figure 5. 3 Prediction comparison of MLE with LSE

Figure 5.3 showed that the prediction performance of the ML estimation method is better than the LS estimation method based on this numerical example. The prediction result of

the ML estimation method for the fault detection process is better compared with the LS estimation method. The summary of the prediction performance based on MRE is given below in Table 5.3. The results of the MRE criterion show that the ML estimation method has better predictive capability.

Table 5. 3 Prediction performance with criterion MRE

MRE criterion	Prediction with the MLE	Prediction with the LSE
Fault detection process	0.026	0.053
Fault correction process	0.027	0.025
Overall MRE	0.026	0.039

From the economic perspective, the further developed the software testing is, the more expensive it becomes, therefore, accurate prediction is quite important and it means a lot to the software manager.

5.2 Monte Carlo Simulation Study

Since the real data for detection and correction process is limited, in order to provide the reader a firm understanding of the prediction performance of the MLE method, we carry out an empirical study by Monte Carlo simulation (Tausworthe and Lyu, 1996a; Tausworthe and Lyu, 1996b).

5.2.1 Simulation Method

To study the performance of the ML estimation method in general, we complete a simulation study. There are several approaches proposed (Lyu, 1996; Tausworthe and Lyu, 1996a) to generate simulation code for the software reliability dataset. An empirical study is given below by Monte Carlo simulation based on the GO model using MATLAB software. The steps carried out are as follows:

1) Generate two sequences of uniformly distributed data $\{u_i, i = 1,\ldots,N\}$ $\{v_i, i = 1,\ldots,N\}$; set *N=300*.
2) Generate the inter-arrival time of the HPP events as $t_i = -\log u_i$, $i = 1,\ldots,N$.
3) From steps 1) and 2), the arrival time of the HPP events can be simulated as $S_0 = 0, S_i = S_{i-1} + t_i, i = 1,\ldots,N$.
4) $s_i = m_d^{-1}(S_i)$, $i = 1,\ldots,N$ is the simulation of NHPP data for fault detection process, where $m_d(t)$ is the MVF of GO model. When the arrival time goes to infinity, the corresponding probability of fault arrival can be negligible (Lyu, 1996).
5) Based on $\lambda_c(t) = \lambda_d(t - \Delta)$, the arrival time of the NHPP data of the correction process can be generated as $s_i' = s_i + t_i', i = 1,\ldots,N$, where the exponentially distributed time delay $\Delta \sim \exp(\mu)$ is simulated as $t_i' = -\log v_i / \mu$, $i = 1,\ldots,N$, and here μ is the parameter for the exponential distribution.

6) For $j = 1,\ldots,k$, $i = 1,\ldots,N$, if $s_i < j$, $d_i = d_i + 1$, if $s_i' < j$, $c_i = c_i + 1$. We choose k=17 for illustrative purpose.

7) $d_j, j = 1,\ldots,k$, and $c_j, j = 1,\ldots,k$ are the cumulative number of detected faults, and number of corrected faults, respectively.

In our simulation, we generate data with exact detection time and correction time, while the data we use for illustration is grouped data. Actually, there is no difference of the prediction performance between grouped data and exact interval data. Wood (1996) did some simulation and found that the predictions from the simulated exact data and the weekly grouped data were essentially identical.

5.2.2 A Simulation Study

As an illustration, by using $\hat{a}$=165, $\hat{b}$=0.12, $\hat{\mu}$=1.6, and *N=300*, several sets of data are then simulated. A comparison is made between the ML estimation predictions and the LS estimation predictions. Based on the simulated datasets of the first 12 weeks, the ML estimates, and the LS estimates are calculated separately. With the estimated parameters, predictions for the next 5 weeks are made. The criterion used to determine the predictive performance is the MRE as defined before.

Simulation of 10 data sets. 10 sets of data are initially simulated to get a general idea of the predictive performance of the two methods. Figure 5.4 shows the results of the Monte Carlo study by plotting the curves of the average value of RE of the predicted values for

the next 5 weeks, based on the ML estimation method, and the LS estimation method separately.

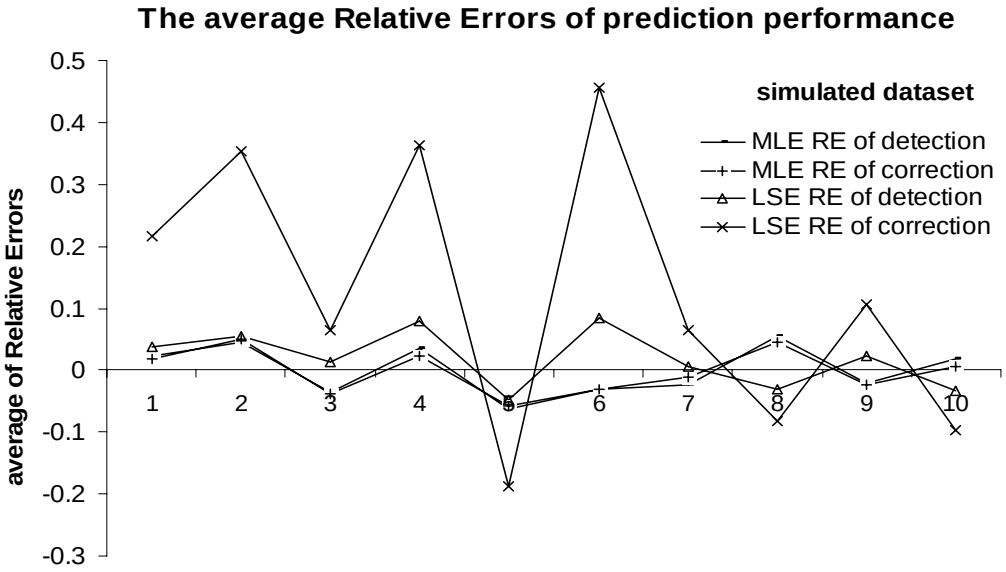


Figure 5. 4 Plot of the average of RE

The simulated datasets were generated with exact detection time and correction time, while the actual dataset used for illustration was grouped data. Some simulations were carried out in (Schneidewind, 2001), and their results showed that the predictions from the simulated exact data were essentially identical to that of the weekly grouped data. The simulation results here show that the ML estimation method had smaller RE compared with the LS estimation method, indicating that the ML estimation method is more stable, and provides narrower variance compared with the LS estimation method.

Simulation of 120 data sets. In order to provide the reader a firm understanding of the prediction performance of the MLE method, in this study, 120 datasets of the fault detection and correction process are simulated with the initial value μ=1.6, μ=2, and

μ=3 separately. MRE is used as a criterion to determine the predictive performance. Here, if the MRE of both detected fault data and corrected fault data of the ML estimation method are less than that of the LS estimation method, the ML estimation method is considered to give better predictions, percentage is used to calculate the percentile of better predictions. If the reverse occurs, the LS estimation method is then considered to give better predictions. In any other case, the two methods are considered incomparable. MRED is used to denote the MRE of detected fault data, and MREC is used to denote the MRE of corrected fault data.

Table 5. 4 The MRE of predicted value simulating 120 datasets

Initial parameter for time delay	MLE	MRED	MREC	%	LSE	MRED	MREC	%
μ=1.6	Mean	0.038	0.034	52.70%	Mean	0.041	0.036	47.30%
	Std.	0.023	0.021		Std.	0.03	0.026	
μ=2	Mean	0.026	0.023	60.20%	Mean	0.028	0.025	39.80%
	Std.	0.022	0.019		Std.	0.025	0.022	
μ=3	Mean	0.027	0.024	72%	Mean	0.037	0.034	28%
	Std.	0.02	0.02		Std.	0.027	0.025	

Table 5.4 shows the mean value, and the standard deviation of MRED, and MREC based on the ML estimation method and the LS estimation method separately. The Monte Carlo simulation results showed that the ML estimation method had a lower value for the mean of MRE for both detection process, and correction process in the prediction performance;

and the standard deviations of MRE for both detection process, and correction process were less than that of the LS estimation method, which means a more stable predictive capability. The results appear to further confirm that the ML estimation method has a more stable predictive capability compared with the LS estimation method, as more than half of the 120 simulated results under three different values of parameter μ show that the ML estimation method gives better prediction compared with the LS estimation method.

The results fall within our expectation. As for MLE method, the idea behind is to determine the parameters that maximize the probability (likelihood) of the sample data. From a statistical point of view, the method is considered to be more robust and yields estimators with good statistical properties, thus may be able to give a better prediction performance. While the LSE method is well known as linear regression, the sum of squares error, and the root means squared deviation is tied to the method. Given a set of data, it may be able to fit the curve very well, but sometimes it can be over-fit and not able to give a better prediction result. LSE might be useful for obtaining a descriptive measure for the purpose of summarizing observed data, but MLE is more suitable for statistical inference.

5.3 Summary

In this chapter, based on traditional NHPP SRGMs, we have proposed a new model considering the fault correction time, and a general framework is proposed to derive a

likelihood function for the combined fault detection and correction process. Based on the new explicit formula, we have showed that MLEs can be easily obtained under different time delay assumptions. An actual set of data from a software development project is presented, different fault correction models are proposed and the ML estimates are calculated given different time delay distributions assumption. Experimental results of the simulation analysis show that the ML estimates with a fairly accurate prediction capability compared with the LS estimates. The approach in our study can be further extended to general SRGMs considering the fault detection and correction process. There are also other researchers proposing their way of prediction analysis. Li et al. (2007) illustrated their experiments in their paper showing that their prediction approach performed quite well in the later stages of software development, and better than single classical software reliability models. They showed that the method could automatically select the most appropriate lower-level model for the data and performances were well in prediction. Existing models typically reply on assumptions about development environments, the nature of software failures and the probability of individual failure occurrences. Thus, each model can be shown to perform well with a specific failure data set, but no model appears to perform well for all cases. In recent years, many methods have been proposed to improve the quality of reliability models. Some nonparametric methods have been introduced into the field, such as Neural Network and Genetic Programming. These types of method are flexible, but they often lack theoretical basis. Further research can be down in this area. Myrtveit et al. (2005) pointed out that empirical study on software prediction models do not converge with respect to the question “which prediction model is best?”. They then did a simulation study, examined a

frequently used research procedure comprising three main ingredients: a single data sample, an accuracy indicator, and cross validation. They found that it was the research procedure itself that was unreliable. Thus, they suggest developing more reliable research procedures before they can have confidence in the conclusions of comparative studies of software prediction models. Further study can be carried out from this perspective of view.

Chapter 6 Optimal Release Time Analysis

As for any traditional SRGMs, the modeling is not the ultimate goal for the analyst. With the incorporation of the fault detection process, more practical information can be extracted, which could be useful to improve the decision-making in a more practical way. One of the important applications of software reliability models is the determination of software release time, which is one of the most important decisions to be made in the software development process. Most of the existing studies on this topic use models based on SRGMs assuming instant fault debugging. In this paper, considering time delay, fault detection and correction modeling analysis is carried out with a new likelihood function derived and the ML estimators obtained. Within this framework, a new economic cost model is proposed considering the time delay. Further analysis on the software release time decision is carried out. This procedure is more reasonable and useful for practical applications. The approach is illustrated with an actual set of data from a software development project.

Based on proposed software reliability models, optimal software release time can be determined by minimizing the total software system cost. In this section, to construct economic models for the total software system cost, several costs that are encountered during software development are reviewed, and various stopping criteria are compared and reviewed. Given the total cost function, and software testing stopping criteria, software cost models can be constructed and optimal release time can be determined. Some traditional software cost models are reviewed considering different optimization problems.

Software reliability is gaining an increasing importance in research and application nowadays. There are many models that have been proposed in the past 20 years (Xie, 1991; Pham and Pham, 2000, 2001; Zhang et al., 2003; Teng and Pham, 2006). As for any traditional SRGM, the modeling is not the ultimate goal for the analyst. The extracted information from the analysis could help the management make decisions regarding the software development project. One of the most important applications of software reliability models is the determination of software release time (Catuneanu et al., 1991; Petrova and Malevris, 1992; Xie and Hong, 1998, 1999; Dohi et al., 1999; Pham and Zhang, a, b, 1999; McDaid and Wilson, 2001; Pham and Wang, 2001; Nishio and Dohi, 2003; Xie and Yang, 2003; Pham, 2003; Berman and Cutler, 2004; Chang and Hung, 2005; Huang, 2005a; Huang and Lyu, 2005a). If a software system is released too early, the user will suffer the failures and great loss; if it is released too late, the competitors will gain competitive advantages.

Okumoto and Goel (1980) first discussed a simple cost model addressing linear developing cost during the testing and operational periods. Ohtera and Yamada (1990) discussed the optimum software-release time problem with fault-detection during operation and introduced two evaluation criteria for the problem: software reliability and mean time between failures. Leung (1992) discussed the optimal software release time given a cost budget. Kapur et al. (1993) discussed software release policies with the optimization criterion minimizing cost subject to achieving a given level of reliability or software performability. Xia et al. (1993) proposed the optimal software release policy with a learning factor for imperfect debugging. Yang and Chao (1995) proposed two

criteria for deciding when to stop testing: the reliability reached and the gain in reliability. Pham (1996) developed a cost model with an imperfect debugging and random life cycle besides a penalty cost to determine the optimal release policies for a software system. Huang and Lo (2006) presented an optimal resource allocation problem in modular software systems during testing phase.

Chari and Hevner (2006) proposed an objective function of total cost, with four cost terms considered:

1) The cost of fixing errors during system testing.
2) The cost of incurring software failures after the software is released.
3) The cost of testing.
4) The cost of delay.

The delay cost, which could represent the cost of missed market opportunity or the penalty cost in meeting the delivery of software in a timely manner, is only incurred when the testing time exceeds the time limit T.

Dohi et al. (2000) pointed out that most existing software release problems have been concerned with the direct minimization of the total expected software cost, but have not taken account of a competitive market situation, which one can observe in the real world.

As traditional SRGMs determined the optimal software release schedule by assuming the stochastic and/or statistical model, Dohi et al. (1999) proposed estimating the optimal software release timing by which minimizing the relevant cost criterion, via artificial neural networks. First, they interpret the underlying cost minimization problem as a graphical one and show that it can be reduced to a simple time series forecasting problem. Secondly, artificial neural networks are used to estimate the fault-detection time in future.

Schneidewind (2005) showed that risk factors have a significant negative effect on reliability. They showed it was feasible to predict risk. They pointed out that although using historical failure data to drive traditional software reliability models would produce greater prediction accuracy, the opportunity to provide early prediction of reliability using risk factors outweighs this advantage.

The optimization model proposed by Pham and Wang (2001) considered the variance of N(t).

$$C(t) = c_3 \cdot t + \left[\frac{2c_0 - c_v}{2}\right] \cdot E(N(t)) + \frac{c_v}{2}\left[Var(N(t)) + E^2[N(t)]\right] \qquad (6.1)$$

Where c_0 as the deterministic part of the cost of fixing software fault; c_v as the random part of the cost of fixing software fault; and c_3 is the cost of testing per unit time. *N(t)* as the cumulative number of software faults detected.

Which assume that the cost of fixing software fault i is a random variable and consists of two parts- deterministic part c_0 and incremental random part c_v.

However, most of those software cost models are based on the assumption that faults detected are corrected immediately. They do not consider the time delay in the fault debugging process and its impact to the optimal release decision policy, while in real life, software managers need to spend some time correcting those faults. Thus, it is more reasonable to assume a time delay between fault detection and fault correction.

Tamura and Yamada (2006) used the coefficient of variation in equation below to assess the reliability requirement, which is a useful measure to decide an optimum time point when the software system has to be released to its operation phase.

$$CV = \frac{\sqrt{Var[N(t)]}}{E[N(t)]} \tag{6.2}$$

Rinsaka and Dohi (2004) considered a more complicated software cost model compared with the simple linear one. They give assumptions that the cost to remove software faults during the testing phase depends on the complexity of the faults, which consists of two parts- the deterministic part and the incremental random part.

They supposed the cost to eliminate the i-th detected software fault could be expressed by

$$C_i = c_4 + (i-1)W, i = 1, \ldots, a \qquad (6.3)$$

where c_4 (>0) is the deterministic cost to remove each software fault during the testing phase; W (>0) is a random variable denoting cost to remove software faults during the testing phase. Cw is the mean value of W. The above assumption can represent an increasing property of fault removal cost in the state *i*. Hence, the expected total cost to detect/ eliminate the software faults during the testing period can be given.

Gokhale (2003) proposed a new cost function considering fault correction process. He considered the cost to customer operations in the field, denoted by c_5, which is a function of the failure rate r(tr) of the software at release time, the expected execution time of the software releases per field site, and the number of field sites. r(tr) is the failure rate of the software in the presence of explicit fault correction. Under the assumption of instantaneous fault correction, r(tr) will be the same as the fault detection rate.

To find a trade-off between risk of releasing too early and the cost of releasing too late, especially considering the time delay, we propose fault detection and correction models with a likelihood function derived to obtain the ML estimators. With incorporation of the fault correction process, more practical information can be extracted, which could be useful in decision-making. Here we investigate the issue of optimal software release policy and propose a new economic model from a new perspective by using the combined fault-detection and fault correction models in this paper.

First, we propose fault detection and correction modeling with a likelihood function derived to obtain the ML estimators. Then several cost factors and cost criteria are reviewed in details. Based on those models considering time delay, given different stopping criteria, we propose new software cost models with more realistic assumptions. Sensitivity analysis is carried out, and some statistical inferences (confidence intervals) are obtained. Further numerical application in optimal release time determination is discussed within our framework.

6.1 Cost Factors and Cost Criteria

6.1.1 Cost Factors

Most of the expected software system cost consists of the following parts.

1) Set-up cost. The initial testing cost which is the barest minimum requirement (Kimura et al., 1999).
2) Testing cost. The expected cost per unit time for testing; the cost to perform testing is assumed to be proportional to the testing time (Pham and Zhang, 1999a).
3) Debugging cost during the testing phase. The expected cost of removing a fault during the testing phase; it is assumed that it takes time to remove each detected fault and the time to remove each fault follows a certain distribution. Gokhale et al. (2006) considered the cost of resolving a failure included the cost of failure identification and fault diagnosis, and the cost of fault removal.

4) Debugging cost during the operation phase. The expected cost of removing a fault during the operation phase; Musa (1993) argued that a consideration of the software's operational profile should reduce system risk. Moreover, it also makes the testing procedure faster and more efficient. Ozekici et al. (2000) proposed a novel model to determine the optimal testing times of software under a given operational profile.
5) The penalty cost. The penalty cost is defined as the cost which should be paid by the manufacturers if the software is delivered after the scheduled delivery time (Tamura and Yamada, 2006).
6) Warranty cost. The maintenance cost per one fault during the warranty period; Kimura et al. (1999) discussed several cases in terms of the behavior of the maintenance cost and assumed that the distribution of the warranty period was a truncated-normal distribution.
7) Risk cost. There is a risk cost associated with the testing coverage. A software provider has to pay each customer a certain amount of money for potential faults in uncovered code (Pham and Zhang, 2003).

6.1.2 Stopping Rules

Software testing is an expensive process, and typically consumes a large part of the cost of the software development project, thus, stopping rules play an important role in constructing software cost models. Many researchers in the literature have addressed the stopping rule problem. Since there are various stopping criteria, Yang and Chao (1995)

compared several stopping criteria which can influence product-release time or product release coverage:

1) Number of remaining faults. Testing can be stopped when a fraction p of the expected total number of faults are detected.
2) Failure intensity requirements. Testing can be stopped when the failure intensity as measured as the end of the test phase reaches a specified value.
3) Reliability requirements. Testing can be stopped when the conditional reliability in the operational phase reaches a required value. Or the ratio of the cumulative number of detected faults at the time T to the expected number of initial faults reaches a specified value (Hou et al., 1997).
4) Availability requirements, the minimum testing time (John and Eamonn, 2005), the loss function of the loss due to testing for one stage (Morali, and Soyer, 2002), and so on.

6.2 Traditional Software Cost Models

As reviewed by Ozekici and Catkan (1993), there are many distinct models in the literature that try to find the optimal stopping time of the testing procedure. Okumoto and Goel (1980) introduced a static optimization model based on the NHPP model for failures. Yamada et al. (1984b) proposed an extension of the G-O model that minimized the total average cost subject to a lower bound on reliability. Bai and Yun (1988) proposed a cost model where the stopping decision was based on the number of the faults corrected

during testing. Hou et al. (1997) considered a model with scheduled delivery times and penalty costs for delayed deliveries. The underlying reliability growth model involves the hyper-geometric distribution. McDaid and Wilson (1997) used a Bayesian decision theoretic approach to identify the optimal release time in a single stage model. That line of research was extended by Morali and Soyer (2002) for the multistage case involving sequential decision making. Huang and Lyu (2005b) proposed optimal release time with emphasize on the cost function for acquiring or developing the automated test tools or new techniques. Without loss of generality, they considered several possibilities of the cost function for developing and acquiring the automated test tools and developed the corresponding optimal software release time. The process of determining whether the software system is ready for release to the next phase is considered as release decision. It involved tradeoffs between continuous testing to increase reliability and prompt release to decrease testing cost. One approach is to use stochastic models that provide measures such as reliability, number of remaining faults, and mean time to failure. Those stochastic models are often extended to include cost as the criterion.

There is a commonly used cost model (Xie, 1991) based on a model with mean value function of *m(t)*:

$$C = c_1 \cdot m(T) + c_2 \cdot \left[m(\infty) - m(T)\right] + c_3 \cdot T \qquad (6.4)$$

in which c_1 is the expected cost of removing a fault during the testing phase; c_2 is the expected cost of removing a fault during the operation phase; c_3 is the expected cost per unit time for testing. Usually, c_2 is at least two orders of magnitude higher than c_1.

The above cost model has its minimum value at

$$T^* = \arg\min_{T \in R^+} \{c_3 \cdot T - (c_2 - c_1) \cdot m(T)\} \tag{6.5}$$

That is equal to find the optimal time T^* with the shortest distance between $c_3 \cdot T$ and $(c_2 - c_1) \cdot m(T)$.

Generally, this optimization problem can be solved numerically. For instance, if the function $m(T)$ is continuous and well-defined, and further is a non-decreasing, strictly concave and bounded function of T, the minimization problem will have a unique solution T^* from $\frac{d^2 m(T)}{dT^2} < 0$ under the assumption that c_2 is much larger than c_1.

For the first time, Zhang and Pham (1998) developed a generalized cost model including fault removal cost, warranty cost, and software risk cost due to software failures. The following cost model calculates the expected total cost:

$$E(T) = C_0 + C_1 T^{\delta} + C_2 m(T) \mu_y + C_3 \mu_W [m(T + T_W) - m(T)] + C_R [1 - R(x \mid T)] \tag{6.6}$$

where C_0 is the set-up cost for software testing; C_1, the software test cost per unit time; C_2, the cost of removing a fault per unit time during testing; C_3, the cost to remove fault detected during the warranty period, usually C_3 is much larger than C_2; C_R, the loss due to software failure; E(T), the expected total cost of a software system at time T; μ_y, the expected time to remove a fault during the testing period; μ_W, the expected time to remove a fault during the warranty period; T_w as the warranty period and $R(x \mid T)$ as reliability of lasting for another x time period. To start from the simplest case, we may assume μ_y and μ_W to be the same. δ is the discount rate of the testing cost.

The above cost model has its minimum value at

$$T^* = \arg\min_{T \in R^+} \left\{ C_1 T^{\delta} + C_3 \mu_W m(T + T_W) - \left[C_3 \mu_W - C_2 \mu_y \right] m(T) - C_R R(x \mid T) \right\} \qquad (6.7)$$

Generally, this optimization problem can be solved numerically. More details are discussed in section 6.6.

6.3 A New Economic Model Considering Time Delay

Usually, faults are assumed to be fixed immediately after detection, that is, the fault removal time is negligible. That is unrealistic and with limited capability, because there is only one process (detection process) and provide only passive information. Within the framework of fault detection and correction modeling as described in chapter 3, based on

the literature review of traditional software cost models, an optimization model is proposed in this section to find a way to give more accurate and useful analysis and decision making, as further information about fault debugging time is included.

6.3.1 Assumptions

Here, new optimization models are proposed based on the following assumptions:

1) Faults detected are not removed immediately, instead, we assume there is a fault debugging time; in this paper, we assume the time to remove each error during the testing period is a random variable with certain distribution under certain conditions.
2) The software application is subject to failures at random times due to the remaining faults within the software system.
3) Usually, the expected cost of removing a fault during the operation phase is at least two orders of magnitude higher than the expected cost of removing a fault during the testing phase.
4) The cost to perform testing is proportional to the testing time.
5) As software is often updated after release in reality. To make more reasonable assumption, here we assume that there is software reliability growth occurring after the testing phase.

6.3.2 The Impact of Time Delay

In traditional SRGMs, $m(t)$ is for the fault detection process, that is, our $m_d(t)$. By considering the impact of time delay, incorporating the fault correction process $m_c(t)$, which is different from the fault detection process, the simple cost model mentioned above (Xie, 1991) can be expressed as

$$C(T) = c_1 \cdot m_c(T) + c_2 \cdot \left[m_d(\infty) - m_c(T)\right] + c_3 \cdot T \tag{6.8}$$

In which $m_c(T)$ is the total number of corrected faults at the time of release T; $m_d(\infty)$-$m_c(T)$ is the number of uncorrected faults that includes two components: undetected faults $m_d(\infty)$-$m_d(T)$ and detected but uncorrected faults $m_d(T)$-$m_c(T)$. By minimizing this cost model with respect to time T, a more practical optimal release time T^* can be calculated. Generally, this optimization problem can be solved numerically. More details are discussed in section 6.6.

Specifically, assuming a simple form considering time delay to be constant, analytical solution can be reached. Based on the G-O model, the corresponding cost function is

$$\begin{aligned} C(T) &= \left(c_1 - c_2\right) \cdot m_c(T) + c_2 \cdot a + c_3 \cdot T \\ &= \begin{cases} c_2 \cdot a + c_3 \cdot T & T \le \Delta \\ \left(c_2 - c_1\right) \cdot a \cdot e^{b\Delta} \cdot e^{-bT} + c_1 \cdot a + c_3 \cdot T & T > \Delta \end{cases} \end{aligned} \tag{6.9}$$

which has its minimum value at

$$T^* = \begin{cases} 0 & \Delta > \Delta' \\ \frac{1}{b} \cdot \ln\left(\frac{c_2 - c_1}{c_3} \cdot ab \right) + \Delta & \Delta < \Delta' \end{cases} \tag{6.10}$$

$$\Delta' = \frac{c_2 - c_1}{c_3} \cdot a - \frac{1}{b} \cdot \left[1 + \ln\left(\frac{c_2 - c_1}{c_3} \cdot ab \right) \right] \tag{6.11}$$

Otherwise, if we do not consider the impact of time delay, then we have

$$\begin{aligned} C(T) &= (c_1 - c_2) \cdot m_d(T) + c_2 \cdot a + c_3 \cdot T \\ &= (c_1 - c_2) \cdot a \cdot \left(1 - e^{-bT}\right) + c_2 \cdot a + c_3 \cdot T \end{aligned} \tag{6.12}$$

which has its minimum value at

$$T^* = \frac{1}{b} \cdot \ln\left(\frac{c_2 - c_1}{c_3} \cdot ab \right) \tag{6.13}$$

Comparing Eq. (6.10) and Eq. (6.13) we can see the impact of time delay to the optimal software release time. Based on above methods, we can further develop new cost models with different time delay distributions (Xie et al., 2007).

6.4 Interpretation of the Cost Parameters

The interpretation of the cost parameters are of great importance, and the structure of the optimal solution depend on their properties. As pointed out by Ozekici et al. (2000), all

economic and stochastic parameters of the model depend on the test case and the specific operation performed. All debugging costs, during testing and after release, depend on the operation that fails. Of course, all cost parameters are non-negative. We also assume that a failure is more costly if it occurs after release other than during testing process.

6.5 Our Generalized Optimization Model

Many existing models assume that the error removal time is negligible, however, in practice both time and money costs are incurred in the isolation of error removal. Also the costs paid to remove these errors are usually quite considerable. Therefore, in order to make the cost model more useful, the time and cost to remove errors and a penalty cost due to the software failure must be addressed in the cost models. Based on the model proposed by Zhang and Pham (1998), considering the fault detection and correction process, assuming a random delay with exponential distribution between fault detection and fault correction, a generalized optimization model can be constructed with several parts of its cost described as below:

Notation

C_0	set-up cost at the beginning of the software development process
C_1	software test cost per unit time
C_2	cost of removing a fault per unit time during testing
C_3	cost to remove fault detected during the warranty period
C_R	the loss due to software failure
$C_1 T^{\alpha}$	cost to perform the testing
$C_2 \cdot \mu \cdot m_c(T)$	cost incurred in removing errors during the testing phase
$C_3 \cdot \mu' \cdot [m_c(T+T_W) - m_c(T)]$	cost incurred in removing errors during the warranty period
$C_R \cdot [1 - R(x \mid (T))]$	risk cost due to software failure
δ	discount rate of the testing cost
Δ	the time period required to remove each error during the testing phase
x	mission time
T_W	warranty period
T^*	optimal software release time
$R(x\|T)$	reliability function of software by time T for a mission time x
$E(T)$	expected total cost of a software system at time T
μ'_y	expected time to remove a fault during the testing period
μ'_W	expected time to remove a fault during warranty period

Thus, the expected total software cost can be expressed as the sum of the above costs $\{E_i(T)\}, i = 0,1,\ldots,4$. The generalized economic cost model considering time delay between FDP and FCP is as below:

$$E(T) = C_0 + C_1 T^{\delta} + C_2 m_c(T)\mu'_y + C_3 \mu'_W \left[m_c(T + T_W) - m_c(T)\right] + C_R\left[1 - R(x \mid T)\right] \qquad (6.14)$$

The cost model has its minimum value at

$$T^* = \arg\min_{T \in R^+} \left\{C_1 T^{\delta} + C_3 \mu'_W m_c(T + T_W) - \left[C_3 \mu'_W - C_2 \mu'_y\right] m_c(T) - C_R R(x \mid T)\right\} \qquad (6.15)$$

Generally, this optimization problem can be solved numerically. More details are discussed in section 6.6. Under different stopping rules, we can obtain different optimal release time. Details are discussed in the next section.

Teng and Pham (2004) considered dividing every item of the software cost model into two items: one is related to testing process, the other is related to the operational process. Similarly, we can consider the reliability during the testing process and the operational process separately; details are shown in the next section.

6.6 The Optimal Release Time

As for any traditional SRGM, the modeling is not the ultimate goal for the analyst. The extracted information from the analysis could help the management make decisions regarding the software development project. With incorporation of the fault debugging time, more practical information can be extracted, which could useful in decision-making.

Based on the proposed economic model, we can obtain the optimal software release time subject to different software testing stopping criteria.

6.6.1 Solution without Constraints

Similar to what Okumoto and Goel (1980), Leung (1992), and Huang and Lyu (2005a) have done, based on the software cost model in Eq. (6.14), assuming no other constraints, some further discussion about the optimal release time is carried out as below.

$$y(T)=\frac{\partial E(T)}{\partial T}=\delta c_1 T^{\delta-1}+c_2\lambda_c(T)\mu_y'+c_3\mu_w'\left[\lambda_c(T+T_W)-\lambda_c(T)\right]-c_4R'(x\mid T) \quad (6.16)$$

$$u(T)=\frac{\partial^2 E(T)}{\partial T^2}=\delta(\delta-1)c_1T^{\delta-2}+c_2\lambda_c'(T)\mu_y'+c_3\mu_w'\left[\lambda_c{}'(T+T_W)-\lambda_c'(T)\right]-c_4R''(x\mid T) \quad (6.17)$$

Given $c_0, c_1, c_2, c_3, c_4, x, \mu_y', \mu_w', T_W$, the optimal value T^* can be obtained under the following conditions:

1. If $u(T)$ is a decreasing function of T, and

   a) If $u(0)\le 0$ and

      (1) if $y(0)\le 0$, then $T^*=\infty$;

      Proof: if $u(0)\le 0$ and $u'(T)\le 0$, we have $u(T)\le 0$ for $T\ge 0$; as $y'(T)=u(T)$ and $y(0)\le 0$, then $y(T)\le 0$ for $T\ge 0$. Since $E'(T)=y(T)$, $E(T)$ is a decreasing

function for all $T \geq 0$, therefore, the minimum of $E(T)$ occurs when $T \to \infty$, which means $T^* = \infty$.

(2) if $y(\infty) > 0$, then T^*=0;

Proof: similar as above, $y'(T) = u(T) \leq 0$ for $T \geq 0$. Then $y(T) > 0$ for $T \geq 0$. Therefore, $E(T)$ is an increasing function for all $T \geq 0$, and the minimum of $E(T)$ occurs at $T = 0$, which means T^*=0.

(3) if $y(0) > 0, y(T) > 0, T \in \left(0, y^{-1}(0)\right]$ and $y(T) < 0, T \in \left(y^{-1}(0), \infty\right)$ then

$$T^* = 0, \text{ if } E(0) \leq E(\infty);$$

$$T^* = \infty, \text{ if } E(0) > E(\infty).$$

Proof: Similarly, $y'(T) = u(T) \leq 0$ for $T \geq 0$. Since $E'(T) = y(T)$, $y(0) > 0$, $y(T) > 0, T \in \left(0, y^{-1}(0)\right]$ and $y(T) < 0, T \in \left(y^{-1}(0), \infty\right)$, $E(T)$ monotonically increases when $T \in \left(0, y^{-1}(0)\right]$ and decreases when $T \in \left(y^{-1}(0), \infty\right)$, then $E(T)$ reaches its maximum at $T = y^{-1}(0)$, and its minimum at $T^* = 0$, if $E(0) \leq E(\infty)$ and $T^* = \infty$, if $E(0) > E(\infty)$.

b) If $u(\infty) > 0$ and

(1) if $y(0) \geq 0$, then T^*=0;

(2) if $y(\infty) < 0$, then $T^* = \infty$;

(3) if $y(0) < 0, y(T) < 0, T \in \left(0, y^{-1}(0)\right]$, $y(T) > 0, T \in \left(y^{-1}(0), \infty\right)$ then $T^* = y^{-1}(0)$;

Proof: Similar as above, so omitted here.

c) If $u(0) > 0, u(T) \geq 0, T \in \left(0, u^{-1}(0)\right)$ and $u(T) < 0, T \in \left(u^{-1}(0), \infty\right)$ then

(1) if $y(0) \geq 0$, then $T^* = 0$;

(2) if $y(0) < 0$, then $T^* = \infty$;

2. If $u(T)$ is an increasing function of T, and

a) If $u(0) > 0$ and

(1) if $y(0) \geq 0$, then $T^* = 0$;

(2) if $y(\infty) < 0$, then $T^* = \infty$;

(3) if $y(0) < 0, y(T) < 0, T \in (0, y^{-1}(0)], \ y(T) > 0, T \in (y^{-1}(0), \infty)$ then $T^* = y^{-1}(0)$;

Proof: Similar as above, so omitted here.

b) If $u(\infty) < 0$ and

(1) if $y(0) \leq 0$, then $T^* = \infty$;

(2) if $y(\infty) > 0$, then $T^* = 0$;

(3) if $y(0) > 0, y(T) > 0, T \in (0, y^{-1}(0)]$ and $y(T) < 0, T \in (y^{-1}(0), \infty)$ then

$T^* = 0$, if $\mathrm{E}(0) \leq E(\infty)$;

$T^* = \infty$, if $\mathrm{E}(0) > E(\infty)$.

c) If $u(0) < 0, u(T) \leq 0, T \in (0, u^{-1}(0))$ and $u(T) > 0, T \in (u^{-1}(0), \infty)$ then

(1) if $y(0) \geq 0$, then

$T^* = 0$, if $\mathrm{E}(0) \leq E(T_b)$;

$T^* = T_b$, if $\mathrm{E}(0) > E(T_b)$, $T_b = \inf\{T : y(T) > 0\}$

(2) if $y(0) < 0, T = y^{-1}(0)$.

As pointed out by Ozekici et al. (2000), once the convexity of the cost function was established, one could use the Kuhn-Tucker conditions to identify a global optimal solution.

6.6.2 Solutions with Constraints

As reviewed above, there could be various constraints combined together with the economic cost model. One of the most common stopping criteria is the reliability criterion. Below we further consider the case of the optimal release time subject to reliability criterion.

By minimizing the total cost function of the economic model with Eq. (6.14) while satisfying the reliability criterion, we have:

$$\begin{aligned} &\min E(T) = C_0 + C_1 T^{\alpha} + C_2 m_c(T)\mu'_y + C_3 \mu'_W \left[m_c(T+T_W) - m_c(T)\right] + C_R\left[1 - R(x \mid T)\right] \\ &s.t. R(x \mid T) \geq R_0 \end{aligned} \tag{6.18}$$

Note that the parameters should already be estimated by the proposed FDP-FCP model. Those cost coefficients can be given based on experience knowledge. To solve the optimization model with constraints, the Lagrange multiplier method can be applied. It is well known that the conditions of Kuhn-Tucker are the most important theoretical results in the field of non-linear programming, and they must be satisfied at any constrained optimum, local or global, of any linear and most non-linear programming problems (Bazaraa et al., 1993). Consequently, the above equation can be simplified associating multiplier λ, and we need to minimize the equation as follows:

$$E'(T) = C_0 + C_1 T^{\alpha} + C_2 m_c(T)\mu'_y + C_3 \mu'_W \left[m_c(T + T_W) - m_c(T)\right] + C_R\left[1 - R(x \mid T)\right] + \lambda\left[R_0 - R(x \mid T)\right] \tag{6.19}$$

Based on the Kuhn-Tucker conditions (KTC), the necessary conditions for a minimum value of above equation are in existence and can be stated as below (Yamada et al., 1995; Nishiwaki et al., 1996):

$$\begin{aligned} &A1: \frac{\partial E'(T)}{\partial T} \geq 0 \\ &A2: T\frac{\partial E'(T)}{\partial T} \geq 0 \\ &A3: \lambda\left[R_0 - R(x \mid T)\right] = 0 \end{aligned} \tag{6.20}$$

Refer to Yang and Xie (2000), there reliability criterion can be testing reliability, $R(x \mid T) = \exp\{-\left[m(x+T) - m(T)\right]\}$ or operational reliability, $R(x \mid T) = \exp\{-\left[\lambda(T)x\right]\}$, denoting the reliability under testing or operational environment separately.

Given the optimal release time T^*, we are able to give the confidence interval of the reliability at the release time (Teng and Pham, 2006). Thus, we can give the lower bound and upper bound of the reliability constraints, which enable us to give the optimal release time with the worse case of software reliability and the better case of software reliability.

Denote by L the likelihood function in general, the Fisher Information for three parameter $\theta = (a, b, \mu)^T$ models of failure detection/correction process can be given by

$$F(\theta) = -E\left[\frac{\partial^2 \log L(\theta)}{\partial\theta \cdot \partial\theta'}\right] = -E\begin{bmatrix} \frac{\partial^2 \log L(\theta)}{\partial a^2} & \frac{\partial^2 \log L(\theta)}{\partial a \partial b} & \frac{\partial^2 \log L(\theta)}{\partial a \partial \mu} \\ \frac{\partial^2 \log L(\theta)}{\partial a \partial b} & \frac{\partial^2 \log L(\theta)}{\partial b^2} & \frac{\partial^2 \log L(\theta)}{\partial b \partial \mu} \\ \frac{\partial^2 \log L(\theta)}{\partial a \partial \mu} & \frac{\partial^2 \log L(\theta)}{\partial b \partial \mu} & \frac{\partial^2 \log L(\theta)}{\partial \mu^2} \end{bmatrix} \tag{6.21}$$

Thus, we have:

$$F = \begin{bmatrix} -E\left[\frac{\partial^2 \ln L}{\partial a^2}\right] & -E\left[\frac{\partial^2 \ln L}{\partial a \partial b}\right] & -E\left[\frac{\partial^2 \ln L}{\partial a \partial \mu}\right] \\ -E\left[\frac{\partial^2 \ln L}{\partial a \partial b}\right] & -E\left[\frac{\partial^2 \ln L}{\partial b^2}\right] & -E\left[\frac{\partial^2 \ln L}{\partial b \partial \mu}\right] \\ -E\left[\frac{\partial^2 \ln L}{\partial a \partial \mu}\right] & -E\left[\frac{\partial^2 \ln L}{\partial b \partial \mu}\right] & -E\left[\frac{\partial^2 \ln L}{\partial \mu^2}\right] \end{bmatrix} \tag{6.22}$$

The large sample asymptotic covariance matrix V of the maximum likelihood estimators for parameters is the inverse of the Fisher information matrix:

$$V = F^{-1} = \begin{bmatrix} Var(\hat{a}) & Cov(\hat{a},\hat{b}) & Cov(\hat{a},\hat{\mu}) \\ Cov(\hat{a},\hat{b}) & Var(\hat{b}) & Cov(\hat{b},\hat{\mu}) \\ Cov(\hat{a},\hat{\mu}) & Cov(\hat{b},\hat{\mu}) & Var(\hat{\mu}) \end{bmatrix} \tag{6.23}$$

The two sided approximate 100α % confidence limits for the parameters can then be obtained in standard way.

If we define a partial derivative vector for reliability *R(x|t)* as

$$\vec{R}(x \mid t) = \left[\frac{\partial R(x \mid t)}{\partial a}, \frac{\partial R(x \mid t)}{\partial b}, \frac{\partial R(x \mid t)}{\partial \mu}\right] \tag{6.24}$$

Then the variance of R(x|t) can be obtained as

$$Var[R(x \mid t)] = \vec{R}(x \mid t) \cdot V \cdot \left(\vec{R}(x \mid t)\right)^T \tag{6.25}$$

Assume the reliability estimation follows the s-normal distribution; then the 95% confidence interval for reliability prediction *R(x|t)* is:

$$[R(x \mid t) - 1.96 \times \sqrt{Var[R(x \mid t)]}, R(x \mid t) + 1.96 \times \sqrt{Var[R(x \mid t)]}] \tag{6.26}$$

6.7 Numerical Example and Sensitivity Analysis

Below based on the dataset from the testing process on a middle-size software project (Table 4.1), we develop our economic cost models based on the G-O model paired with exponential time delay. The cost coefficients in the cost model are assumed to be known already, as they are usually determined by empirical data, previous experiences, and the nature of the applications (Pham, 2003). The other model parameters are estimated using MLE method based on our proposed likelihood function. Optimal release time can be determined subject to various stopping criteria. Sensitivity analysis is carried out and the impact of time delay on optimal release time is analyzed.

6.7.1 A Simple Cost Model Considering Time Delay

For our numerical example, the analysis is illustrated as follows. Based on the simple cost model with time delay in Eq. (6.8) we have: $\hat{a} = 158$, $\hat{b} = 0.14$ and $\hat{\mu} = 0.64$. Assuming c_1=*\$300*, c_2=*\$500,* c_3=*\$100,* it has its optimal point as T^*=28.83. The cost function for

this data set with our model is plotted in solid line in Figure 6.1, with an apparent minimum point. However, traditional analysis with single fault detection model will lead to different result. Comparatively, against the fault detection data set, the G-O model has estimates as $\hat{a} = 154.21$ and $\hat{b} = 0.1408$, and then the corresponding cost function can be reached as plotted in dashed in Figure 6.1, which has an earlier optimal point as $T^* = 26.78$. The difference between these two results shows the effects of correction time over cost.

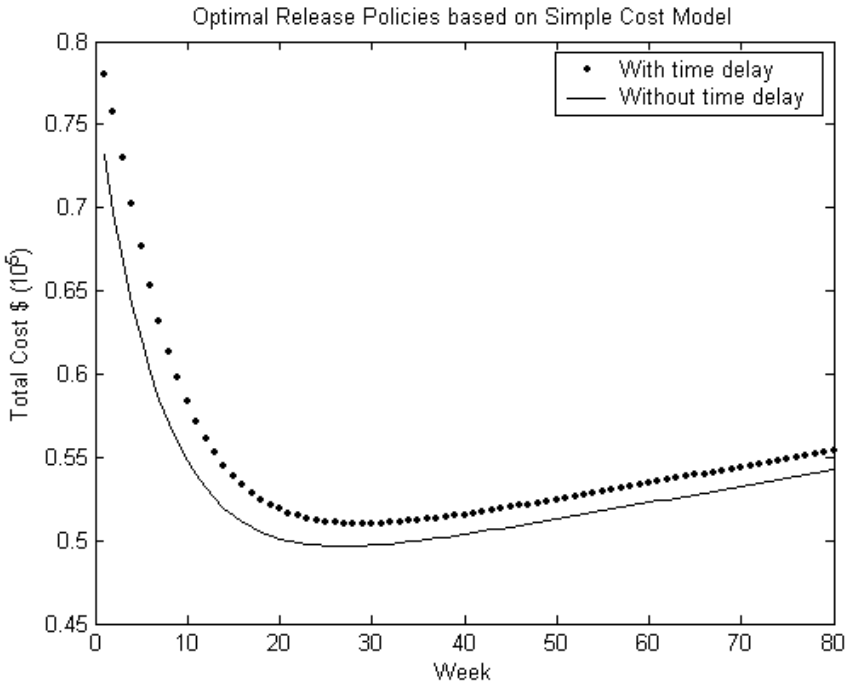


Figure 6. 1 Plot of the total cost functions of a simple cost model

6.7.2 A Generalized Cost Model Considering Time Delay

1) Cost model without constraints

Based on our proposed economic model in Eq. (6.14) considering the time delay, we have $\hat{a} = 158$, $\hat{b} = 0.14$ and $\hat{\mu} = 0.64$. Assuming C_0 *=$50,* c_1*=$700,* c_2*=$60,* c_3*=$3600,*

C_R =*$5000*. To start from the simplest case, we assume μ_y' and μ_W' to be the same as $\mu_y = \mu_W = \mu_y' = \mu_W' = 1.7$, $x = 1, T_W = 20$, α =1. Without considering the time delay function, the optimal release time is T^*=*37.67* with total cost spent as *$ 47344.57*. On the other hand, if considering the time delay between FDP and FCP, we have the corresponding optimal release time as T^*=39.48 with total cost spent as *$ 48611.03*.The corresponding cost functions are plotted out in Figure 6.2. The difference between these two results shows the effects of correction time over cost.

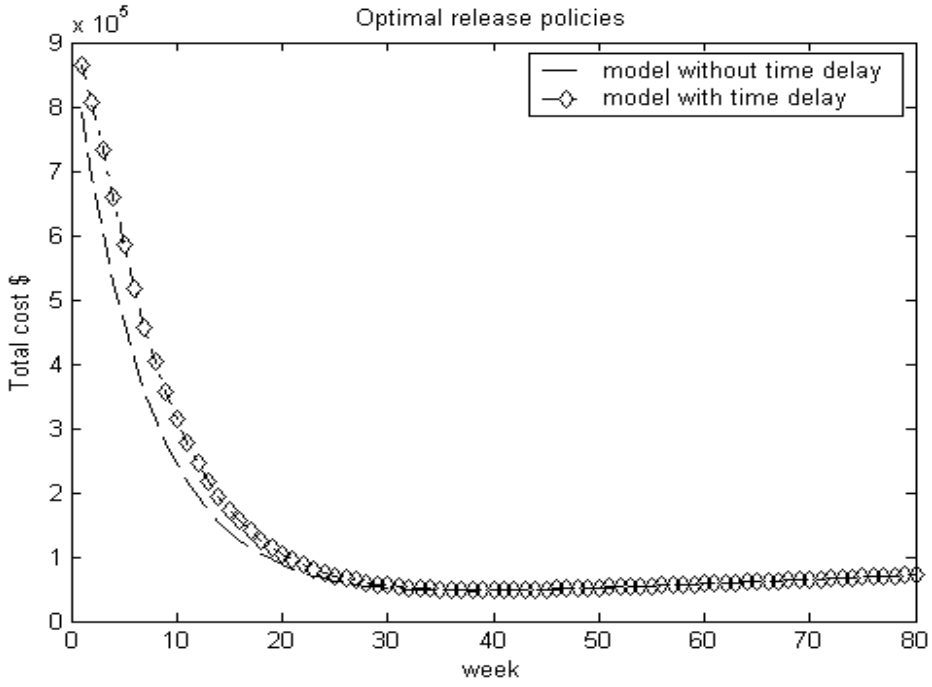


Figure 6. 2 Plot of the total cost functions

2) Cost model with reliability constraints

Based on the results above, we consider the proposed total software cost function subject to various reliability constraints. Below we consider two constraints: testing reliability and operational reliability.

Case 1: Cost model with testing reliability criterion

First we use testing reliability (Yang and Xie, 2000) as the criterion. $\hat{a} = 158$, $\hat{b} = 0.14$ and $\hat{\mu} = 0.64$, assume C_0 =*\$50,* c_1=*\$700,* c_2=*\$60,* c_3=*\$3600,* C_R =*5000,* $\mu_y = \mu_W = \mu'_y = \mu'_W = 1.7$, $x = 1, T_W = 20$, α =1, R_0 =0.95 . The optimal release time with time delay considered is T^*=*42.66* with total cost spent as *\$ 49039.65*.

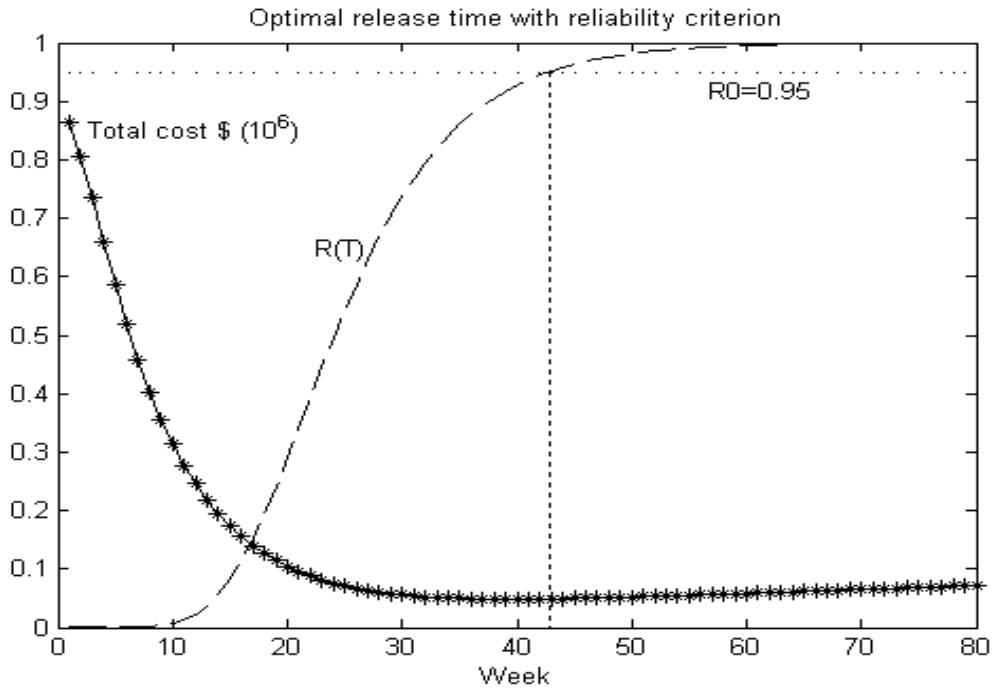


Figure 6. 3 Plot of the total cost functions with testing reliability criterion constraint

Figure 6.3 showed the influence of constraints on the optimal release time. With testing reliability criterion, the optimal release time is delayed compared with unconstrained case. It is easily understood as you need to spend more money to ensure higher reliability,.

Given the optimal release time T^*, we are able to give the confidence interval of the reliability at the release time (Teng and Pham, 2006). With the help of MATLAB, we finally get the variance matrix

$$Var\left(\hat{\theta}\right) = I^{-1}\left(\hat{\theta}\right) = \begin{bmatrix} 355.8016 & -0.3227 & -0.1169 \\ -0.3227 & 0.0004 & 0.0002 \\ -0.1169 & 0.0002 & 0.0346 \end{bmatrix} \tag{6.31}$$

Then the variance of $R(x \mid T)$ when *x=1, T=42.66* can be obtained as

$$\begin{aligned} & Var\left[R(x \mid T)\right] = \vec{R}(x \mid T) \cdot V \cdot \left(\vec{R}(x \mid T)\right)^T \\ & = \begin{bmatrix} \frac{\partial \hat{R}}{\partial \hat{a}} & \frac{\partial \hat{R}}{\partial \hat{b}} \end{bmatrix} \cdot V \cdot \begin{bmatrix} \frac{\partial \hat{R}}{\partial \hat{a}} & \frac{\partial \hat{R}}{\partial \hat{b}} \end{bmatrix}^T \\ & = \begin{bmatrix} -0.00031 & 1.755697 \end{bmatrix} \begin{bmatrix} 355.8016 & -0.3227 \\ -0.3227 & 0.0004 \end{bmatrix} \begin{bmatrix} -0.00031 & 1.755697 \end{bmatrix}^T \\ & = \begin{bmatrix} -0.67686 & 0.000802 \end{bmatrix} \cdot \begin{bmatrix} -0.00031 & 1.755697 \end{bmatrix}^T \\ & = 0.001618 \end{aligned} \tag{6.32}$$

Assume the reliability estimation follows the s-normal distribution; also it is known that $R \in [0,1]$, then the 95% confidence interval for reliability prediction $R(x \mid T)$ is: [0.87, 1].

Case 2: Cost model with operational reliability criterion

Then, we use operational reliability as the criterion, considering the time delay function the optimal release time is T^*=43.15 with total cost spent as *$ 49171.05*.

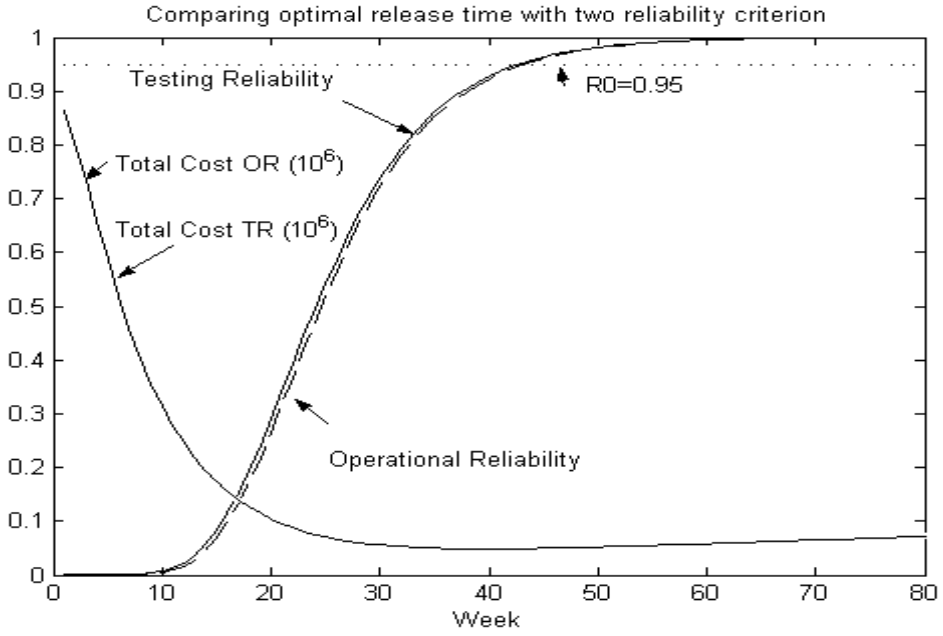


Figure 6. 4 Plot of the two total cost functions with two reliability criteria

Then the variance of $R(x \mid T)$ when *x=1, T=43.15* can be obtained as

$$
\begin{aligned}
&Var\left[R(x \mid T)\right] = \vec{R}(x \mid T) \cdot V \cdot \left(\vec{R}(x \mid T)\right)^T \\
&= \left[\frac{\partial \hat{R}}{\partial \hat{a}} \quad \frac{\partial \hat{R}}{\partial \hat{b}}\right] \cdot V \cdot \left[\frac{\partial \hat{R}}{\partial \hat{a}} \quad \frac{\partial \hat{R}}{\partial \hat{b}}\right]^T \\
&= \left[-0.00031 \quad 1.756751\right] \begin{bmatrix} 355.8016 & -0.3227 \\ -0.3227 & 0.0004 \end{bmatrix} \left[-0.00031 \quad 1.756751\right]^T \qquad (6.33) \\
&= \left[-0.67788 \quad 0.000803\right] \cdot \left[-0.00031 \quad 1.756751\right]^T \\
&= 0.001623
\end{aligned}
$$

Assume the reliability estimation follows the s-normal distribution; also it is known that $R \in [0,1]$, then the 95% confidence interval for reliability prediction $R(x \mid t)$ is [0.87, 1].

We can see that the optimal release problem should be formulated according to the operational reliability criterion since the testing reliability constraint can lead to an incorrect value of the testing time. Figure 6.4 also shows the difference between the testing times of unconstrained and constrained optimization problems. It is clear that for ensuring the operational reliability of the software, more cost is required.

On the whole, from above numerical illustration, we find out that the operational reliability constraint should be adopted instead of the testing reliability while adding a reliability constraint to the software release problem; and the software manager should be aware of the more expensive cost for ensuring the required reliability of the software.

Below, a different parameter estimation method is used to see if there is any effect on the optimal software release time. Considering the case with no constraints, and assuming the same initial values of *c1=$300, c2=$500, c3=$100,* and exponential time delays, the LS estimation method is applied, and the estimated parameters are obtained as $\hat{a} = 156, \hat{b} = 0.14, \hat{\mu} = 0.58$. The optimal release time is T^*=28.95, with a total cost of *$50409.39*. The two optimal release policies are plotted in Figure 6.5.

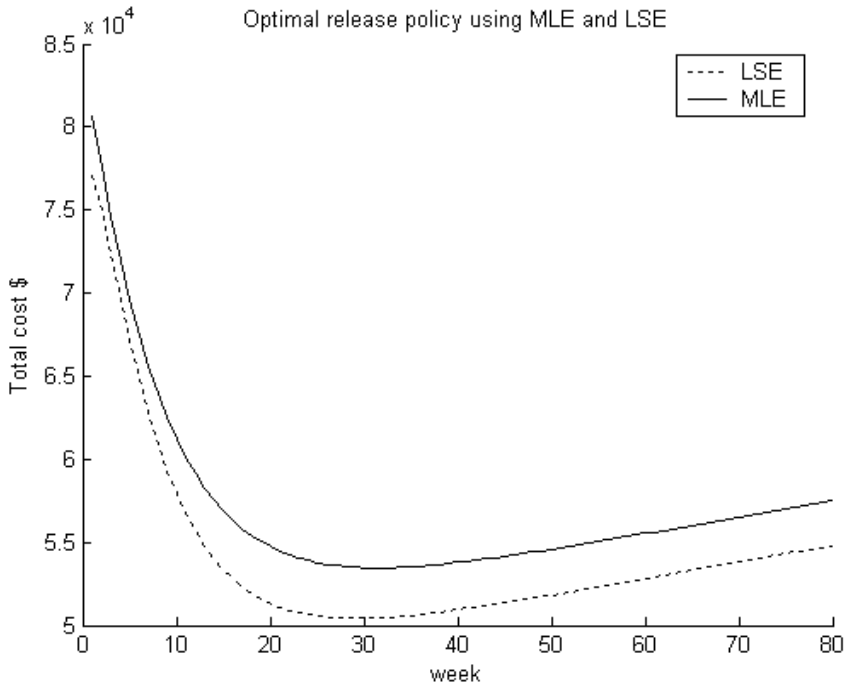


Figure 6. 5 Plot of the total cost function comparing the MLE, and the LSE

From Figure 6.5, the optimal release times for the LS estimation model, and the ML estimation model are T^* =28.95, and T^* =31.29 respectively. It is well-known that software testing accounts for a substantial portion of software development costs, however, releasing software with unacceptable reliability is also very costly. As we known that LSE has no basis for constructing confidence intervals and testing hypothesis, to take a conservative point of view, software manager might adopt ML estimates as it is more suitable for statistical inference, and it can give more accurate information on the impact of the parameter changes. Therefore, in this case, it might be better to release the software later at the optimal time of T^*=31.29 as estimated by the ML method instead of T^*=28.95 as estimated by the LS method. By adopting the conservative estimates of

software release time, it will ensure the software's reliability and quality with good stable performance, and it can also avoid the huge risk cost after it is released.

6.7.3 Impact of the Factors

The impacts of the cost coefficients on the simple cost model with time delay are analyzed below. Similar analysis with more complex models and constraints can be carried out in the same way.

1) The impact of the cost coefficients

The testing cost per unit time is expected to be very low in many applications compared with costs associated with discovering and fixing bugs during testing/operational phase. As it is expected that unreliable software will yield a high loss if it is released, the costs associated with discovering and fixing bugs is usually assumed to be high. Morali and Soyer (2002) showed it was important to note that the decision of when to stop testing may be sensitive to the choice of above parameters. Thus, it is always desirable to investigate the sensitivity of the results to the choice of prior parameters and loss function components.

The impacts of the cost coefficients: c_1, c_2 and c_3 on the expected total cost are analyzed below. Changing the values of the corresponding cost coefficients, the optimal release

time can be changed correspondingly. Below we increase the values of c_1, c_2 or c_3 and keep the values of other parameters unchanged.

Increasing the cost coefficient of removing faults during testing phase, c_1 , from *$ 300* to *$ 450*, we find that the optimal release time changes from T^* = 27.61 to T^* = 18.29, which can be interpreted as the impact of putting weights on cost during testing phrase. The optimal release time is shortened because of high testing cost.

Increasing the cost coefficient of removing faults during operation phase, c_2, from *$ 500* to *$ 750*, we find that the optimal release time changes from T^* = 27.61 to T^* =33.07, which can be interpreted as the impact of putting weights on cost during operation phrase. The optimal release time is delayed because of high operation cost, thus, it needs to maker sure there is as few faults as possible within the software.

Increasing the cost coefficient of software test cost per unit time, *c3* , from *$ 100* to *$ 250*, we find that the optimal release time changes from T^* = 27.61 to T^* =21.45, which can be interpreted as the impact of putting weights on software test cost per unit time. The optimal release time is shortened because of high test cost per unit time.

Increasing the cost coefficient of the loss due to software failure, C_R , from *$ 5000* to *$ 9000*, we find that the optimal release time changes from T^* = 39.48 to T^* =39.90, which can be interpreted as the impact of putting weights on the loss due to software failure.

The optimal release time is delayed because of high risk cost of software failure. The similar analysis can be carried out comparing the other cost coefficients as well.

2) The impact of the warranty time T_W

Considering the impact of the warranty time T_w on the total cost function by increasing T_w from 20 weeks to 50 weeks, we find that the optimal release time changes from T^* = 39.48 to T^* =40.29, which can be interpreted as the impact of putting weights on the warranty time. The optimal release time is increased because of high risk cost of software failure is increased due to longer warranty time.

3) The impact of the mission time *x*

Considering the impact of the mission time *x* on the total cost function by increasing *x* from 1 week to 5 weeks, we find that the optimal release time changes from T^* = 39.48 to T^* =40.56, which can be interpreted as the impact of putting weights on the mission time. The optimal release time is increased because of high risk cost if the mission to lasting longer time is failed.

6.7.4 Interval Estimation of Parameters in the Cost Model

The interval estimation of release time is recommended to avoid further excessive adjustment of release time. We investigate the variation of the optimum release time due

to the variation of the estimated parameters. Usually, the software can only be released when the reliability level has reached a predetermined level, which is usually the customer requirement. For illustrative purpose, here we consider the case without any constraints, the simplest case. We calculate the Fisher information matrix, that is, the matrix of negative second partial derivatives of the log likelihood function, to obtain the asymptotic variances and covariance of the ML estimates of the parameters *a, b* and μ .

Example

Using the same dataset above, with the help of MATLAB, we can get the variance matrix as below:

$$Var\left(\hat{\theta}\right) = I^{-1}\left(\hat{\theta}\right) = \begin{bmatrix} 355.8016 & -0.3227 & -0.1169 \\ -0.3227 & 0.0004 & 0.0002 \\ -0.1169 & 0.0002 & 0.0346 \end{bmatrix} \tag{6.34}$$

Given $\alpha = 0.05$, we can then give 95% confidence interval of those MLE estimated parameters. For example, from the Fisher information matrix, we get $Var(\hat{a}) = 355.8016$, thus, the 95% confidence interval of the parameter $\hat{a}$ with estimated value of 165.37 is [128.3991, 202.3409]. In the similar way, we can give interval estimation of other parameters in the software cost model as well.

6.7.5 Sensitivity Analysis of Optimal Release Time

The sensitivity issue of software release time can be further studied, that is, the variations of the optimum release time due to the variation of the estimated parameters. If an

overestimation of a parameter implies an underestimation of the release time which can be costly as more failures are experienced by the consumers, we should try not to overestimate the parameter (Xie and Hong, 1998). Also, if a parameter affects the release time more than others, it is important to have this parameter estimated as accurately as possible.

The sensitivity issue can be very complex, and it varies among different release time models. Thus, here we can restrict it to the simple case, that is, the optimal release policy by minimizing the cost only.

Denoting model parameters as variable $\theta = (a, b, \mu)^T$, then we have:

$$m_c(T,\theta) = a \cdot \left[1 - \frac{\mu}{\mu - b} e^{-bT} + \frac{b}{\mu - b} e^{-\mu T} \right]$$

$$E(T,\theta) = C_0 + C_1 T^{\alpha} + C_2 m_c(T,\theta)\mu'_y + C_3 \mu'_W \left[m_c(T + T_W) - m_c(T), \theta \right] + C_R \left[1 - R(x \mid T) \right] \qquad (6.35)$$

with $\frac{\partial E(T,\theta)}{\partial a} > 0$ for all T>0, which implies that $E(T,a)$ is an increasing function of a. If we overestimate parameter a, then we will overestimate the optimal release time; on the other hand, if we underestimate parameter a, then we will underestimate the optimal release time as well. We can analysis other parameters in a similar way. If $\frac{\partial E(T,\theta)}{\partial b} < 0$ (or $\frac{\partial E(T,\theta)}{\partial \mu} < 0$), we should try not to overestimate that parameter so as to reduce the probability that customers experiencing more failures. The sensitivity analysis result can help us better allocate resources for a more accurate estimation for the most

important parameters. It also provides a way to obtain reasonably conservative estimate of the release time.

Example

Assume all other parameters are given, changing parameter a from 150 to 160, we can see the total cost increase $ 1287.91; in the similar way, if changing parameter b from 0.14 to 0.15, we can see the total cost decrease $ 1238.8, if changing μ from 0.58 to 0.59, the total cost amount decrease $ 23.81. Comparing the above three, we can see parameter b is of the most important, while parameter μ is of less importance. In this case, the sensitivity analysis result indicates that it is important to have the parameter b estimated as accurately as possible, then parameter a and parameter μ. Different time delay function and software cost function can give different results; therefore, it is important to carry out the sensitivity analysis to help software managers have a better idea of all those model parameters and their impact.

6.8 Summary

In this paper, fault detection and correction modeling analysis is carried out with a new likelihood function derived and the ML estimators obtained. Within this framework, an economic model based on FDP and FCP is proposed, and the optimal release policies considering the time delay are analyzed in details. Many assumptions are relaxed in this cost model, fault debugging time is considered and the warranty and risk cost issues are included. The proposed new economic model can provide more accurate results and give

a more reasonable rationalizing measure to make a better decision of software release policy. Software managers can obtain the corresponding optimal release time when the mission time being changed, the warranty period shortened or prolonged, or any other factors changing as well. Further studies can be done by taking into account imperfect debugging so as to make more realistic assumptions. As the parameters of the failure process and costs are dependent on the operations that the software performs, this can be a future direction for further research.

Chapter 7 Bayesian networks modeling for software inspection effectiveness

Except for testing, the only other widely applicable technique for detecting and eliminating software defects is to review and walkthrough during inspection process. As removing faults during inspection process is much cheaper compared with testing process, we consider finding as much errors as possible during the inspection process. Since inspection effectiveness is considered as an important criterion to judge the inspection performance, our concern is to construct a model to measure the inspection effectiveness. However, software inspection process is flexible and complex. There is no unified inspection structure and there are many factors contributing to its effectiveness for each specific procedure. That motivates us to use Bayesian network models to measure the inspection effectiveness.

Software inspection is a cost-effective approach to detect and remove defects from software in the early phase of software development lifecycle. In order to control the software inspection process, some related measurements are required. As reviewed before, there have been many different approaches to measure software inspection effectiveness. Unfortunately, these natural but simplistic measurement definitions regard software inspection as a mechanical process. In fact, software inspection process is flexible and complex, with no unified structure. Many contributing factors are highly dependent on the experience of individual inspectors, introducing great uncertainty into this process.

Starting from this point, in this chapter, a Bayesian networks model has been proposed to describe the interdependencies within inspection structure and the contribution of each factor to the overall belief on inspection effectiveness (Cockram, 2001). We propose our Bayesian network model based on the one given by Cockram (2001). Cockram proposed a BN model which could help improving the inspection performance. This model is interesting and it provides a framework to use Bayesian networks to develop uncertain reasoning over inspection effectiveness. However, there are some shortcomings with this model, which it failed to incorporate the critical information reflecting the status of the inspection process, i.e., the remaining number of faults. Inspections are developed to deduct the faults left in software artifact, and this measurement directly denotes the effect of the inspection. Cockram (2001) did not incorporate this factor by arguing that those quantities were not available at the time of inspection and may never be known unless the execution of the software caused the errors to be manifested as faults, In fact, many research work have been devoted in evaluating inspection effectiveness with respect to this measurement through subjective or objective estimation approaches. Naturally, this estimated variable would influence the belief over the inspection effectiveness, and it should be regarded as collected evidence successively updated over the whole inspection process.

By adding the variable of remaining number of faults, our evaluation on the effectiveness can be updated with new collected data, keeping the inspection process under track. Besides, we also propose a systematic method to establish the prior belief of those nodes so as to initialize the Bayesian network. We propose two methods to obtain those prior

probabilities. The first method is given through calculating the pair-wise matrix using Expert Choice software so as to give the prior belief of those root parent nodes in the BN. The second method is given using Best-Fit software to find out the best-fit distribution for the normalized data value and finally give the a-priori conditional probability table.

In this chapter, further investigation on modeling software inspection effectiveness through Bayesian networks is carried out. Specifically, the former two shortcomings are compensated with proposed approach. The rest part of this chapter is organized as follows. Section 7.1 proposes the network structure and the systematic knowledge extraction approach. In section 7.2, the proposed model and probability extraction approach is illustrated with a numerical example. Also, related sensitivity analysis is developed.

7.1 Software Inspection Process

Software has become a central part in any complex system, and software of quality has become a common requirement. However, to develop software satisfying this requirement within the constraints of budget and schedule is still a challenging problem. Software inspection has been generally accepted in software development as a cost-effective approach for quality improvement through defect removal (Aurum et al., 2002). Such a static verification technique is originally introduced in Fagan (1976), and has been studied and applied extensively with many varieties (Kelly and Shepard, 2004b; Miller and Yin, 2004). Generally speaking, it is a systematic technique to examine any software

artifact for defect detection and removal, and can be applied to the early phase in software development.

In order to control the software inspection process, some related measurements are required. There have been many different attempts to measure software inspection effectiveness. As inspection is to remove software defects, it is natural to use the related defects number as the measurement. Gilb and Graham (1993) gave a definition of inspection efficiency as:

$$\text{Inspection efficiency} = \frac{\text{Number of defects}}{\text{Cost consumed by inspections}} \tag{7.1}$$

Some works suggest using the already detected defects to calculate the measurement, i.e., defect density (Porter et al., 1997; Perry et al., 2002). This measurement actually denotes the efficiency of the developed inspection, and the remaining defects number seem to be a better alternative as it denotes the effect of the inspection on the software. Both objective and subjective approaches have been taken to develop estimation on this measurement (Biffl, 2003). Capture-recapture is a well studied approach to develop related estimation (Emam and Laitenberger, 2001; Petersson et al., 2004). However, it is criticized with the extra cost and difficulties added in defect implantation, and some alternatives are developed through the time series trend or subjective judgments on the collected data (Yin et al., 2004; Amasaki et al., 2005). In order to evaluate the effectiveness of the software inspections, we surveyed the literature and find there are

various evaluation of inspection effectiveness, such as inspection efficiency, return on investment, and cost-effectiveness.

Besides the definition given above by Gilb and Graham (1993), Collofello and Woodfield (1989) defined the economic impact of inspections in terms of the ratio between the cost and the benefit measured as effort saved.

$$\text{Inspection\_effectiveness} = \frac{\text{Cost saved by inspection}}{\text{Cost consumed by inspection}} \tag{7.2}$$

Franz and Shih (1994), Grady and Slack (1994) and Rico (2004) defined the economic impact of inspections in terms of return-on-investment (ROI) as:

$$\text{ROI} = \frac{\text{Cost saved by inspection - Cost consumed}}{\text{Cost consumed by inspection}} \tag{7.3}$$

Kusumoto et al. (1992) defined inspection cost-effectiveness as:

$$\text{Inspection effectiveness} = \frac{\text{Cost saved by inspection - Cost consumed}}{\text{Potential defect cost without inspection}} \tag{7.4}$$

In this paper, our definition of the inspection effectiveness is based on the third one that is the definition proposed by Kusumoto et al. (1992).

These natural but simplistic measurement definitions regard software inspection as a mechanical process. However, software inspection process is flexible and complex. There is no unified inspection structure and there are many factors contributing to its effectiveness for each specific procedure (Biffl and Halling, 2003; Briand et al., 2004). Many of these factors are highly dependent on the experience of individual inspectors, introducing great uncertainty into this process (Kelly and Shepard, 2004a; Perry et al., 2002). Kollanus (2005) introduced several problems in inspection practices, such as meeting scheduling may cause delay, meetings consume resources with few gains in finding new defects, and participants do not understand inspection process, and then he gave solutions to those problems so as to improve the inspection. However, there are still many other factors that could influence the inspection process and the inspection effectiveness too. Aurum et al. (2005) investigated the inspection effectiveness with altering some of the inspection attributes, such as the environmental context, document type and reading technique. Freimut et al. (2005) proposed a model to measure inspection cost-effectiveness and a method to determine the cost-effectiveness by combining project data and expert opinion. However, those models can only give analysis to a certain few attributes. When we consider more and more attributes that influence the inspection effectiveness, it would be quite difficult to analyze using their methods.

Starting from this point, a Bayesian networks model has been proposed to describe the interdependencies within inspection structure and the contribution of each factor to the overall belief on inspection effectiveness (Cockram, 2001). This model is interesting and it provides a framework to use Bayesian networks to develop uncertain reasoning over

inspection effectiveness. However, there are some shortcomings with this model. Firstly, the network excludes the factor of remaining number of faults, which is actually an important measurement denoting the inspection effects on the software. Secondly, no systematic approach other than brainstorming is developed to extract knowledge from experts, and this brings more uncertainty and possible inconsistency into this modeling framework. Thirdly, it measures the inspection effectiveness in a static way. Software inspection is a dynamic process, and the updating of some information could cause the belief change, such as the detected number of defects. Although the importance of inspectors' learning process is emphasized, its dynamic influence on the inspection effectiveness is not well explored. Herein, further investigations on modeling software inspection effectiveness through Bayesian networks are carried out.

7.2 Bayesian Networks

The definitions of a Bayesian network can be found in many versions, and the basic form by (Pearl, 1986) is stated as follows: Bayesian network is a directed acyclic probability graph, connecting the relative variables with arcs, and this kind of connection expresses the conditional dependence between the variables.

In Bayesian networks, variables are used to express the events or objects. The problem could be modeled with study on the behavior of these variables. In general, we first calculate (or determine from expert experience) the probability distribution of each variable and the CPD (Conditional Probability Distribution) between them and the

probability distributions of the root parents. Then with these distributions we can obtain the joint distributions of these variables. Finally, some deduction can be developed for some variables of interest with some other variables known.

To understand the approach of Bayesian networks, a simple example is shown here. As shown in Figure 7.1, this Bayesian network models height relationships among three members of a family: father, mother and son, which are denoted by the notes F, M and S respectively. Their casual connections are obvious: the height of the son is influenced by both the mother and father, which are expressed with the directed arcs in Figure 7.1.

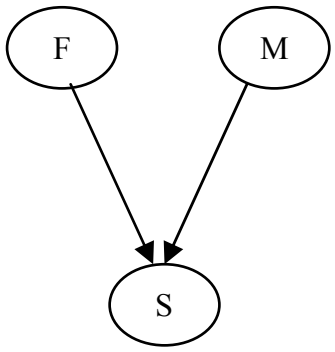


Figure 7. 1 A simple example of Bayesian network

Accordingly, in the example there are three random variables, F, M, and S, each of which is the height defined as tall (1) and short (0). Their relationships as shown in the figure are expressed by CPD. As F and M do not have parent nodes, their probability distributions can be defined as (Pr(F=1), Pr(F=0)) and (Pr(M=1),Pr(M=0)), with Pr() expressing the probability. Node S has parents as F and M, so its CPD (Pr(S| F,M)) are dependent on various combinations of F and M. To illustrate the problem, we assume the

related distributions as follows. Let Pr(F=1) =0.5, Pr(F=0) =0.5; Pr(M=1) =0.5, Pr(M=0) =0.5, and Pr(S| F,M) is shown in Table 7.1.

Table 7. 1 CPD of node S

Pr(S| F,M)	S=1	S=0
F=1, M=1	0.7	0.3
F=0, M=1	0.5	0.5
F=1, M=0	0.5	0.5
F=0, M=0	0.3	0.7

Following this, some inferences can be developed based on this Bayesian network with structure and relationships known.

Due to these desirable properties, Bayesian networks have been increasingly applied in many fields, including software engineering. It has been used to predict software reliability in the early phases of the development by incorporating information ahead of testing (Smidts, 1998), to develop a causal model for software defect rates prediction (Fenton, 1999), and to manage software project risk (Fan and Yu, 2004).

The Bayesian network has been proposed to model the software inspection process. The first step in Bayesian networks modeling is to identify the contributing variables and their inter-dependencies, i.e., to identify the network structure. This involves clearly description of the inspection process structure and investigation with experienced inspectors.

To model the inspection process of software, first of all, we need to construct a network and store in each node a conditional probability distribution of the variable, conditioned on the outcome of all uncertain variables that are parents of that code. The initialization of a Bayesian network requires that the priori belief in terms of the conditional probability for each state of the variables in the parent nodes be specified. Experience is used to provide a priori conditional probability value for each node matrix. For the root parent nodes, which are at the bottom of the network, the initial distribution for each state of these variables is set to be flat over its ranges, i.e. the evidence has an equal probability for each state. For each node within the network the initial belief must be established as a probability potential (conditional probability table).

7.3 Model Development

7.3.1 Bayesian Network Framework

To identify the contributing variables and their inter-dependencies is the first step in Bayesian networks modeling, and it involves clearly description of the inspection process structure and investigation with experienced inspectors.

Generally, the contributing variables can be divided into three groups: inspection structure factors, artifacts under inspection, and related inspection proceeding status data. Firstly, software inspection has no unified procedure and many variations have been evolved ever since Fagan's basic method. As a result, different Bayesian networks structure should be developed for different inspection structures. Without loss of

generality, we take Fagan's basic structure as illustration (Fagan, 1986). Secondly, the software artifact under inspection has great influence on the effectiveness, both its size and complexity. The Bayesian networks dealing with these two groups have been proposed with a clearly defined hierarchical network structure (Cockram, 2001) for static inspection analysis.

However, that model fails to incorporate the critical information reflecting the status of the inspection process, i.e., the remaining number of faults. Inspections are developed to deduct the faults left in software artifact, and this measurement directly denotes the effect of the inspection. Cockram (2001) did not incorporate this factor by arguing that those quantities were not available at the time of inspection and may never be known unless the execution of the software causes the errors to be manifested as faults. However, as denoted earlier, many research works have been devoted to evaluating the inspection effectiveness with respect to this measurement through subjective or objective estimation approaches. Naturally, this estimated variable variant would influence the belief over the inspection effectiveness, and it can be regarded as collected evidence successively updated over the entire inspection process. By adding this variable, our evaluation on the effectiveness could be updated with new collected data, keeping the inspection process under track.

To illustrate the modeling approach described in the former section, a numerical example is developed in this section. The related analysis is developed with the aid of NETICA software of version 2005. Below Figure 7.2 is the Bayesian network we proposed, based

on the one proposed by Cockram (2001), and refer to those BBNs proposed by Fan and Yu (2004) and Laitenberger and Baud (2000).

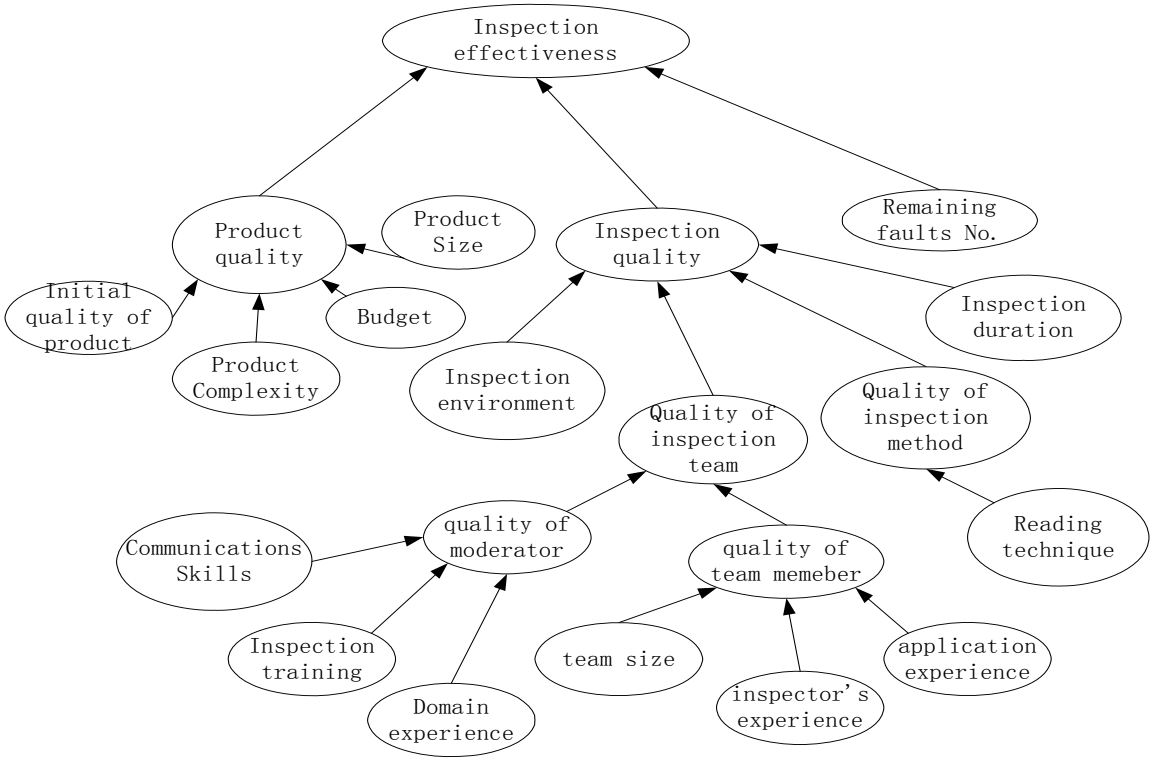


Figure 7. 2 A proposed Bayesian network model

7.3.2 Bayesian Network Configuration

With the established Bayesian network model, we have the qualitative interdependencies between these identified variables. However, in order to develop inference, further quantitative dependencies need to be identified, i.e., the conditional probability distribution over the dependent nodes and the probability distribution of the root parents.

Prior conditional probability distributions. Some questionnaires based on the brain-storming of the inspectors have been used to generate these probabilities in (Cockram, 2001). Freimut et al. (2005) proposed using a triangular distribution to explicit the expert opinion. That is, the expert is asked to provide a range, given by minimum and maximum values, in which the estimate can be, and the most likely values.

Prior conditional probability distributions for the intermediate nodes. Some questionnaires based on the brain-storming of the inspectors have been used to generate these probabilities in the paper by Cockram (2001). Freimut et al. (2005) proposed using a triangular distribution to explicit the expert opinion. That is, the expert was asked to provide the most likely values, and a range in which the estimate could be, given by minimum and maximum values.

An alternative approach is to assume the conditional distribution to follow some specific distribution, and we can use the maximum likelihood estimation to fit the data obtained from expert to find out the best fit distribution and get the corresponding parameters. As a result, Most of the best-fit results of the data analysis tends to show that the conditional probability distribution in the inspection process is a general Beta distribution with pdf (probability density function) defined in [0, 1] as

$$f(x \mid \alpha, \beta) = \frac{(1-x)^{\beta-1} x^{\alpha-1}}{B(\alpha, \beta)} \tag{7.5}$$

in which $B(\cdot)$ is the beta function and $\alpha, \beta > 0$.

Prior probability distributions static nodes. The rest root parents' probability distributions are highly human-oriented and have to be extracted from experts' knowledge. Cockram (2001) set those distributions to be uniformly distributed. Such a scheme would waste the prior experience from inspectors. To develop model for any specific inspection, it is favorable to extract the prior knowledge from experienced inspectors. Rosqvist et al. (2003) proposed a method to encode the experts' tacit knowledge into probabilistic measures associated with the achievement level of software quality attributes. The author argued that a software expert (developer or assessor) is capable of expressing his opinion on the achievement level of a quality attribute based on the mental model of the software.

Here considering the risk of inconsistent probability elicitation, mathematics-based AHP method is proposed to elicit the consistent probabilities from experts. AHP is based on the pair-wise judgments of the importance of the different attributes of interest, and then a priority ordering of these attributes can be derived, with a measure of inconsistency. Firstly, for each root parent node, all the possible values are given as its attributes, such as "poor, fair, and good". Then with respect to this variable, the pair-wise judgments are generated on these values, and the priority is given as the distribution of this variable over these values. The mathematics related to AHP provides a rule for consistency checking, and it provides a systematic approach for prior belief elicitation.

As we have emphasized, "remaining number of faults" and "inspectors' experience" change over the inspection process, while the left factors keep static relatively. For those

root parent nodes which may change as time goes by, regarding them as dynamic nodes we can use a Discrete Time Markov Chain (DTMC) model to model this learning process. The parameters used in the modeling can be abstracted from former projects. Ergodic DTMCs have a stable distribution after a warm-up process, which can be used to describe the evolvement of those dynamic root parent nodes within the BN modeling.

Among these static factors, "preparation time", "team size", "formal actions" and "exit criteria" are specific and can be measured in a general way. "Product size" and "product complexity" are common software metrics, and they can be measured with the artifact under inspection (Fenton, 1999). The rest root parents' probability distributions are highly human-oriented and have to be extracted from experts' knowledge. In (Cockram, 2001), these distributions are set to be uniform, and such a scheme would waste the prior experience from inspectors. To develop model for any specific inspection, it is favorable to extract the prior knowledge from experienced inspectors. Rosqvist et al. (2003) proposed a method to encode the experts' tacit knowledge into probabilistic measures associated with the achievement level of software quality attributes. The author argued that a software expert (developer or assessor) is capable of expressing his opinion on the achievement level of a quality attribute based on the mental model of the software. In our paper, considering the risk of inconsistent probability elicitation, mathematics-based AHP method (Saaty, 1980) is proposed to elicit the consistent probabilities from experts. AHP is based on the pair-wise judgments of the importance of the different attributes of interest, and then a priority ordering of these attributes can be derived, with a measure of inconsistency. Firstly, for each root parent, all the possible values are given as its

attributes, such as “poor, fair, and good”. Then with respect to this variable, the pair-wise judgments are generated on these values, and then the priority is then regarded as the distribution of this variable over these values. The mathematics related to AHP provides a rule for consistency checking, and it provides a systematic approach for prior belief elicitation.

Prior probability distributions of dynamic factors. “Remaining number of faults” and “inspectors’ experience” are two dynamic factors in software inspection process. During inspection, each detected fault is recorded and submitted for correction. Although the new faults could be introduced during correction, the number of remaining faults should show a decreasing trend in the long run. The estimation of this can be developed through a subjective approach (Emam et al., 2000; Yin et al., 2004). Combining the AHP approach we proposed, this approach is developed as follows. The method is as below: We need an inspector to provide a subjective estimate of his/her effectiveness, which will be denoted as $\hat{E}$. Therefore, if the inspector estimates that 75% of the defects in a document were found, then the value of $\hat{E}$ would be 0.75. Then, if the detected faults No. is d, then we have d/ $\hat{E}$ as the total number of faults. Therefore, the remaining fault count should be d/ $\hat{E}$ -d.

Such an estimation procedure can be developed after a fixed period of time, such as one week. Then the distribution is updated in the Bayesian network to estimate the inspection effectiveness successfully. With the experts’ experience increasing, the subjective estimation would become more and more convincing. The estimation before the

inspection can be developed by referring to inspection results on similar software project (Xie et al., 1999).

"Inspectors' experience" also changes over inspection process, and the evolution denotes the learning process of the inspectors. We can use the similar approach to developed estimation, but to use inspectors' experience to generate the distribution f "inspectors' experience" would have confliction inside. It is hard for an inexperienced inspector to make judgments over his own experience level. Therefore, we propose to use a discrete time Markov chain (DTMC) to model this learning process, and the parameters of the model can be abstracted from former projects. Ergodic DTMCs have a stable distribution after a warm-up process, which could be used to describe the evolvement of the inspectors' experience well.

7.4 Numerical Example

To illustrate the modeling approach described in the former section, a numerical example is developed in this section. The related analysis is developed with the aid of decision-analysis software, explained as below in detail.

7.4.1 Bayesian Network Modeling

Same as the network structure in Figure 7.2, the Bayesian network can be constructed with the help of software tools. However, due to the size of the network and the limitation

of space, the following example is illustrated with part of the network as shown in Figure 7.3.

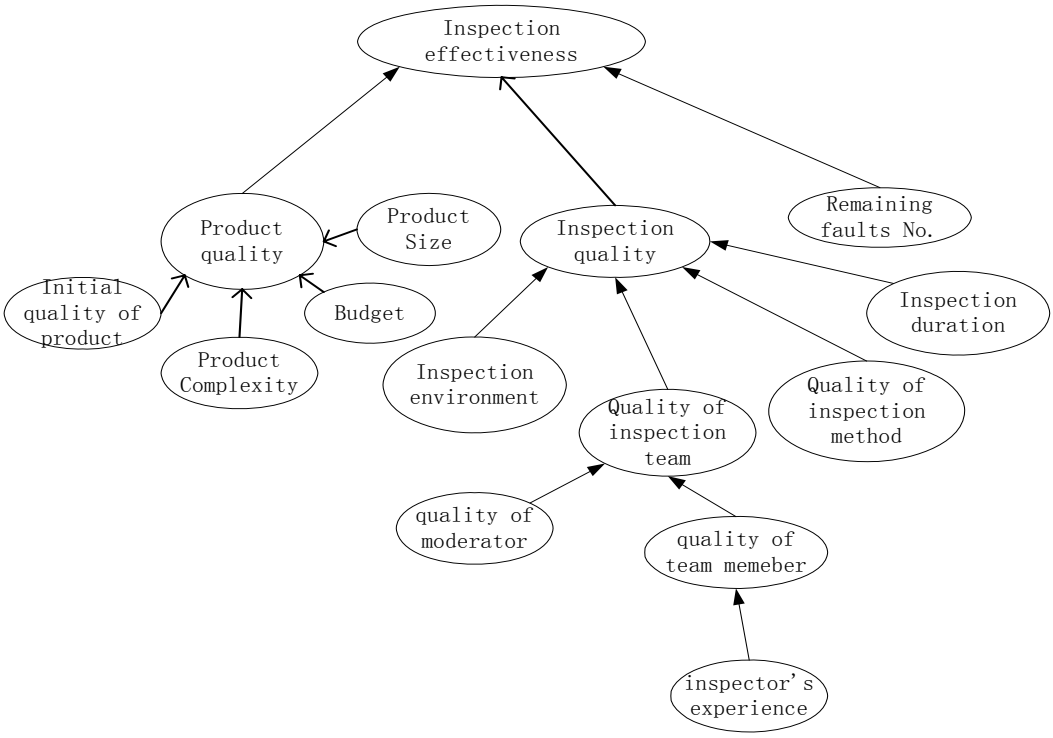


Figure 7. 3 Part of Bayesian network model

7.4.2 Networks Probability Distributions

Prior conditional probability table. Through investigation with software inspectors, a large number of data are collected related to the conditional distributions. These data are normalized into the value interval [0, 1], and we use maximum likelihood estimation within the BEST-FIT software to find out the best-fit distribution for those data. Most of them turn out to be a Beta distribution.

Table 7. 2 Prior CPD of inspection effectiveness over inspection quality

Inspection effectiveness	very poor	poor	medium	good	very good
Inspection quality	9	9	3	0	0

For example, we knew that the conditional probability distribution of inspection effectiveness depends on various combinations of its four parent nodes. Here assuming that the other three nodes are of medium state, we then obtained the conditional probability distribution of the inspection effectiveness conditioning on the quality of the inspection. We asked 21 experts to feedback on the inspection effectiveness (such as poor, medium and good) given the inspection quality; the corresponding data is shown in Table 7.2. The Pearson-Tukey three-point approximation is used for estimating the probability distribution for each parameter whose variation is represented, which was proposed (Keefer and Bodily, 1983) as the first of the discrete-distribution approximations. The Pearson-Tukey approximation suggested using the 5, 50, and 95 percentiles to get the probabilities of the approximate p.m.f. as 0.185, 0.630, and 0.185. We applied this method to give the proper value of those attributes of each node. For example, we can use this method to define the value of poor, medium and good as value of x, where f(x) is the 5, 50, and 95 percentiles of the p.d.f. function.

For example, we know that the conditional probability distribution of inspection effectiveness depends on various combinations of its four parent nodes. Here assuming that the other three nodes are of medium state, we can obtain the conditional probability distribution of the inspection effectiveness conditioning on the quality of the inspection. The best fit distribution turned out to be Beta-General (0.124, 0.126, 0, 9). By using that

method, we could give the conditional probability table to construct the Bayesian network for software inspection.

Prior probability distribution. In order to estimate the prior probability distribution, we need to ask the expert to give the pair-wise comparison matrix. In practice, the inconsistency always occurs. As a compromise, we need to check whether the matrix is acceptable or not. If not, we should ask the expert to re-judge the probability, and redo it until we get the acceptable matrix. Then we will use AHP to get the corresponding sets of weights. Thus, prior probability distributions of root parent nodes are obtained to be used in Bayesian network.

For example, we have the expert opinion to evaluate the variable of initial quality of product. Then, we use the Expert Choice software (Expert Choice, 2005) to treat it as the Analytic Hierarchy Process (AHP). This variable is supposed to take attributes as poor, fair, and good, and then we ask the expert to give their opinion with their experiences as in Table 7.3. For example, the likelihood that the initial quality is poor vs. the likelihood that the initial quality is fair is 4/3.

Table 7. 3 Pair-wise comparison matrix for the node "initial quality of product"

	Poor	Fair	Good
Poor	1	4/3	4/3
Fair	3/4	1	1
Good	3/4	1	1

The priorities are obtained, i.e., the distribution w = [*0.4, 0.3, 0.3*]. After that, we need to check for the inconsistency of the matrix. Fortunately, we find that it was a perfectly consistent matrix, and we take this as the belief for this variable.

In our proposed BN model, we assume that the nodes "remaining number of faults" and "inspectors' experience" change over the inspection process, with the rest factors kept static relatively. Among these static factors, "preparation time", "team size", "formal actions" and "exit criteria" are specific and can be measured in a general way. "Product size" and "product complexity" are common software metrics, and they can be measured with the artifact under inspection (Fenton, 1999).

"Remaining number of faults" and "inspectors' experience" are two dynamic factors in software inspection process. During inspection, each detected fault is recorded and submitted for correction. Although the new faults could be introduced during correction, the number of remaining faults should show a decreasing trend in the long run. The estimation of this can be developed through a subjective approach (Emam et al., 2000; Yin et al., 2004). Combining the AHP approach we proposed, this approach is developed as follows. We need an inspector to provide a subjective estimate of his/her effectiveness, which will be denoted as $\hat{E}$. Therefore, if the inspector estimates that 75% of the defects in a document were found, then the value of $\hat{E}$ would be 0.75. Then, if the detected number of faults is d, then we have d/$\hat{E}$ as the total number of faults. Therefore, the remaining number of faults should be d/$\hat{E}$-d. Such an estimation procedure can be developed after a fixed period of time, such as one week. Then the distribution is updated

in the Bayesian network to estimate the inspection effectiveness successfully. With the experts' experience increasing, the subjective estimation would become more and more convincing. The estimation before the inspection can be developed by referring to inspection results on similar software project (Xie et al., 1999).

"Inspectors' experience" also changes over inspection process, and the evolution denotes the learning process of the inspectors. We can use the similar approach to develop estimation, but to use inspectors' experience to generate the distribution of "inspectors' experience" would have confliction inside. It is hard for an inexperienced inspector to make judgments over his own experience level. Therefore, a DTMC can be used to model this kind of nodes. Similarly, the probability distributions for the other human-oriented variables can be gathered to configure the Bayesian network model.

7.4.3 Model Analysis

With all the Bayesian network structure, the related conditional probability, and the probabilities, some inferences can be developed for further insight to evaluate the inspection effectiveness. Specifically, we are interested in exploring two related properties: the dynamic changes of the inspection effectiveness with the process proceeding, and the sensitivity analysis to find out variables contributing most to the inspection effectiveness.

The basic inference is to follow the direction of networks: with the available conditional/unconditional probability distributions and the collected metrics, it is straightforward to deduct the distribution of the unknown variable of inspection effectiveness as shown in Figure 7.4 using NETICA software (because of the limitation of nodes No. within NETICA software, we use part of the BN model for illustration).

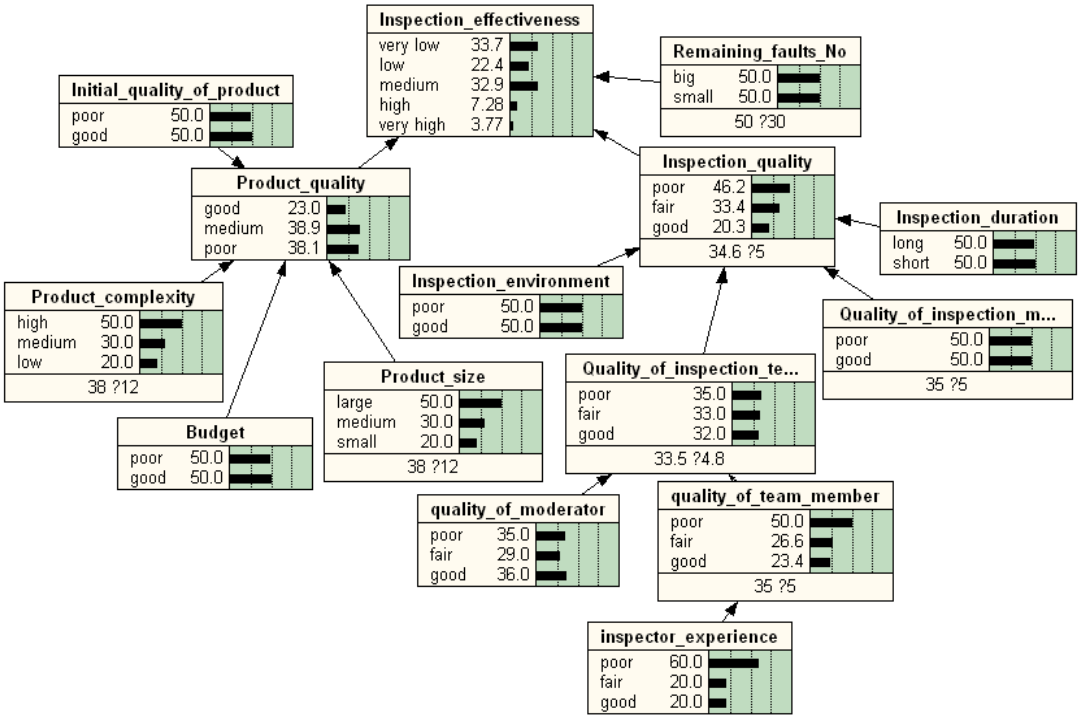


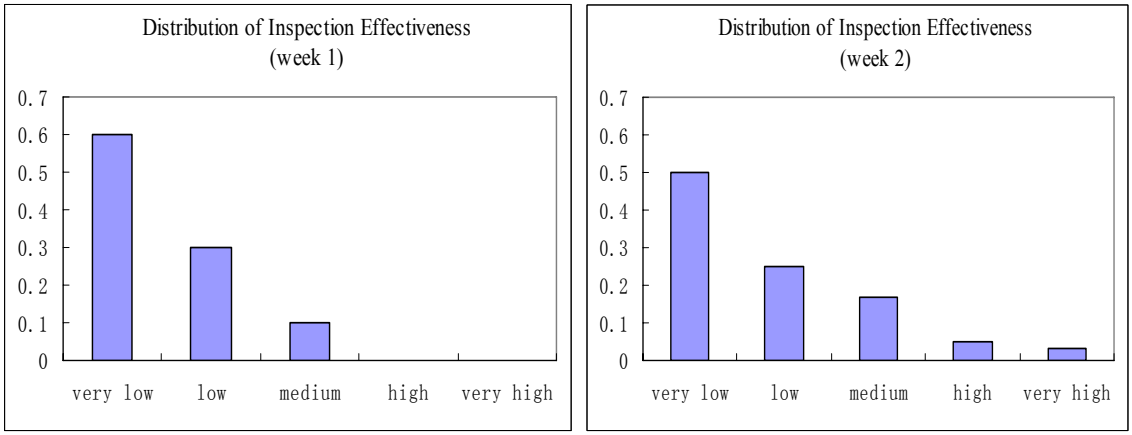
Figure 7. 4 Numerical Example of BBN (part of the BN model)

We find that the parent nodes are static except the node of remaining number of faults and the node of inspector experience. As a result, the inspection effectiveness should be evaluated dynamically: updated with each new collected belief over the remaining fault count and over the inspector experience. Such a scheme has the advantage of evaluating the effectiveness of inspection along the process running, providing nearly on-line feedback on the status of software inspection.

7.4.4 Dynamic Analysis of the Node "Remaining number of faults"

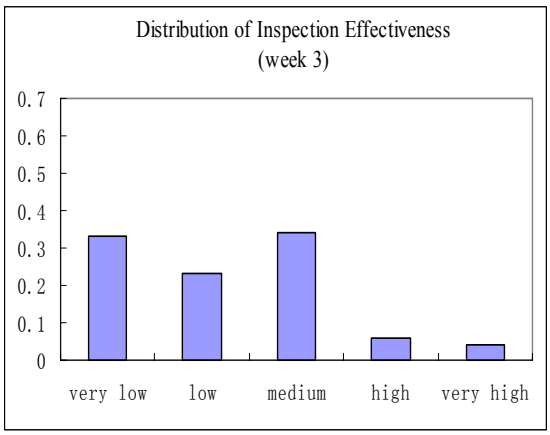
Below we give a dynamic analysis of inspection effectiveness over remaining number of faults. To show the improvement of the inspection effectiveness over the decreasing of remaining fault count, we consider the case assuming that the percentile of remaining fault count of big over small is 100/0, 80/20, 50/50, 20/80, 0/100 for five weeks consequently. That is, we assume that the No. of big faults remaining is decreasing over time, and we give the assumption that the percentile of remaining big faults is 1, 0.8, 0.5, 0.2, and 0 for week 1, 2, 3, 4, 5 so as to carry a dynamic analysis in a more convenient way.

The deducted believes over inspection effectiveness are described in Figure 7.5, showing the dynamics of inspection effectiveness with the ongoing process, assuming all other nodes in the BN model unchanged. We can see the belief over the inspection effectiveness increases with the decreasing of the belief of the number of remaining faults in the software. That is an example to show how to conduct dynamic analysis. We can use the similar way to analyze each node that is dynamic and human related, and we can plot out the change of its probability distribution and see the corresponding change of the inspection effectiveness.

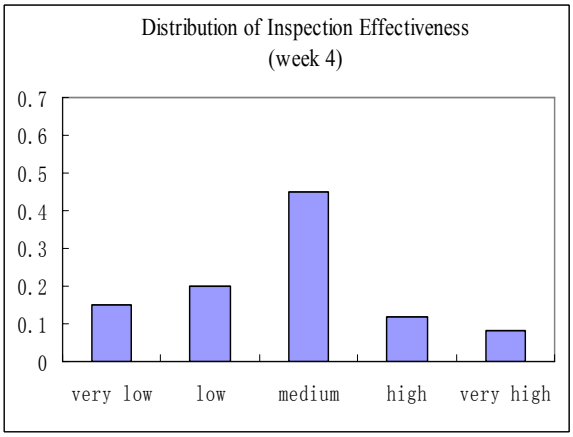


(1)

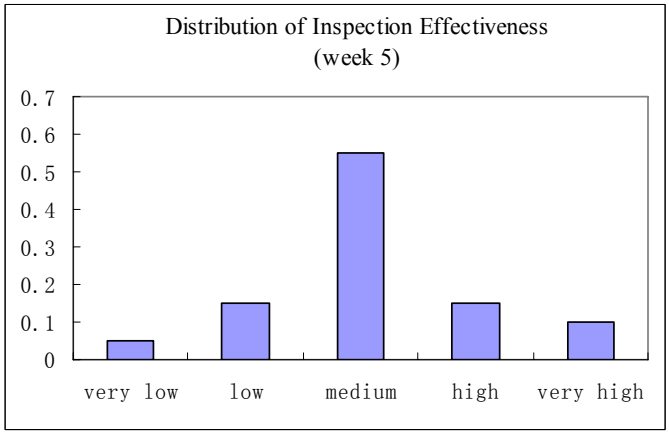
(2)



(3)



(4)



(5)

Figure 7. 5 Inspection effectiveness changes with respect to remaining number of faults

Apart from such direct analysis, Bayesian networks inference can be developed from any direction. Therefore, it is convenient and flexible to develop sensitivity analysis. NETICA software is used to carry out a sensitivity analysis for this Bayesian network. We only consider part of network due to the limitation of the version of the NETICA software we used. The sensitivity analysis is to examine the corresponding change of the network "output" with the change of some "input". With the help of NETICA, the analysis procedure is illustrated with an example as shown below in Figure 7.6; we give the dynamic analysis of the node "inspection effectiveness" by changing the node "inspection experience". Below we only plot out a BN within 15 nodes due to the limitation of the NETICA software.

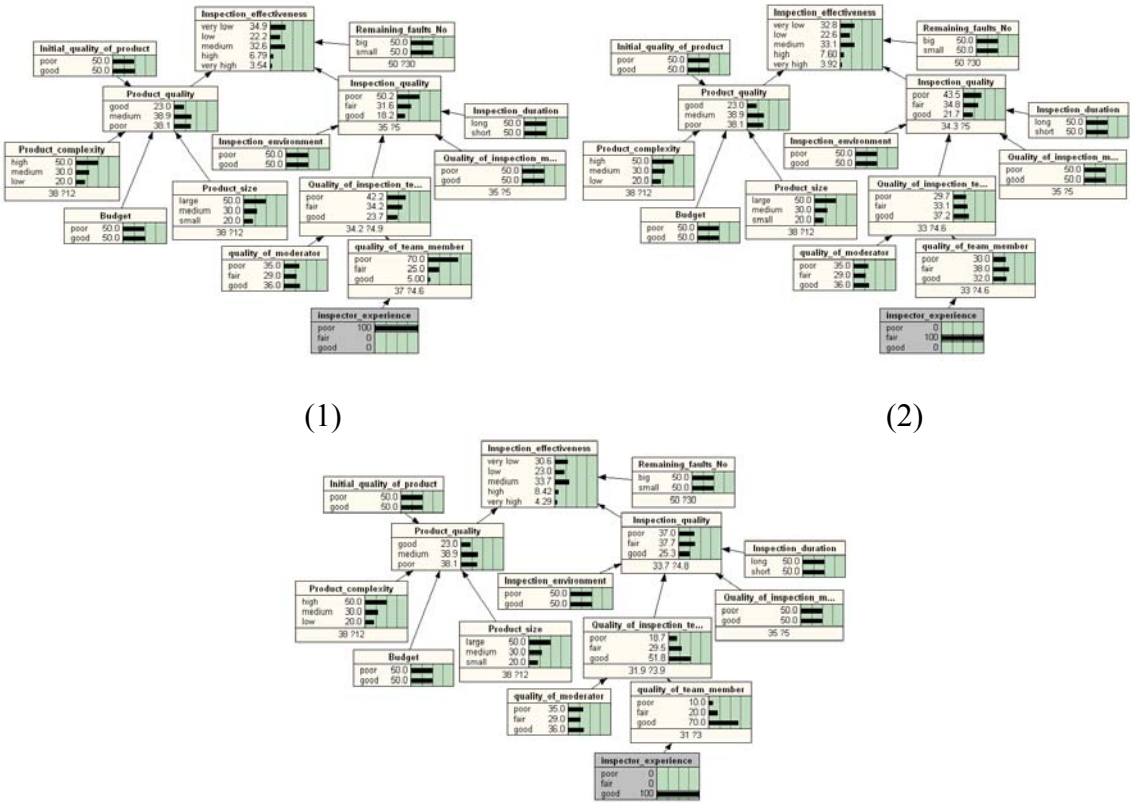
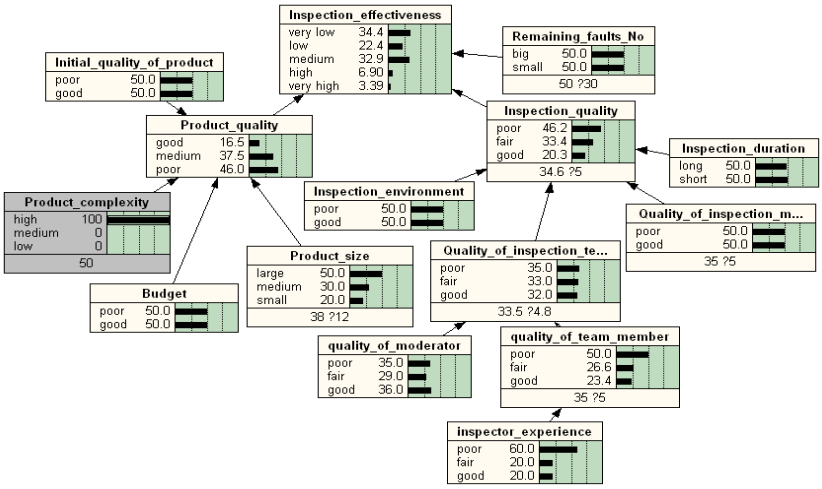


Figure 7. 6 Corresponding change of other nodes while change the sate of the node "inspector's experience"

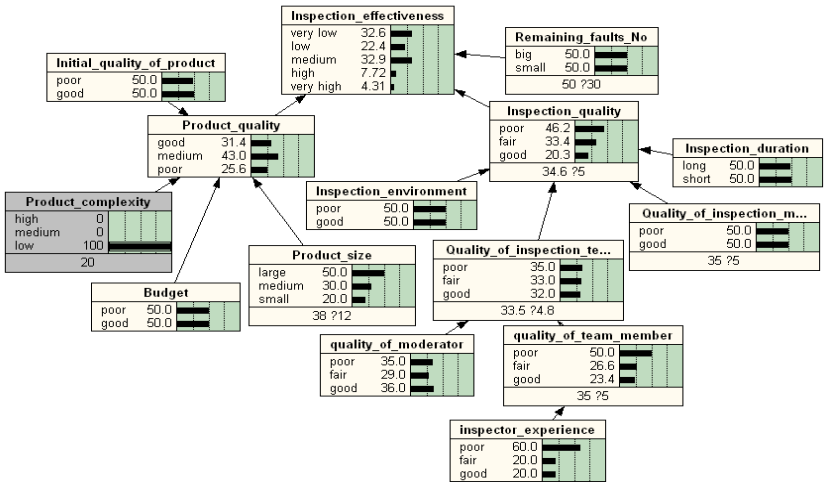
We can use the similar way to analyze each node that is dynamic and human related, and we can plot out the change of its probability distribution and see the corresponding change of the inspection effectiveness. With all the Bayesian network structure, the related conditional probability, and the probabilities, some inferences can be developed for further insight to evaluate the inspection effectiveness. Specifically, we are interested to explore another related property: the variables contributing more to the inspection effectiveness.

7.4.5 Sensitivity Analysis

Apart from such direct analysis, Bayesian networks inference can be developed through any direction. Therefore, it is convenient and flexible to develop sensitivity analysis. NETICA (NETICA 2005) is used to carry out a sensitivity analysis for this Bayesian Network. The sensitivity analysis is to examine the corresponding change of the network "output" with the change of some "input". With the help of NETICA, the analysis procedure is illustrated with the following Figures 7.7, Figure 7.8, and Figure 7.9. We could change some factors and see the corresponding changes.

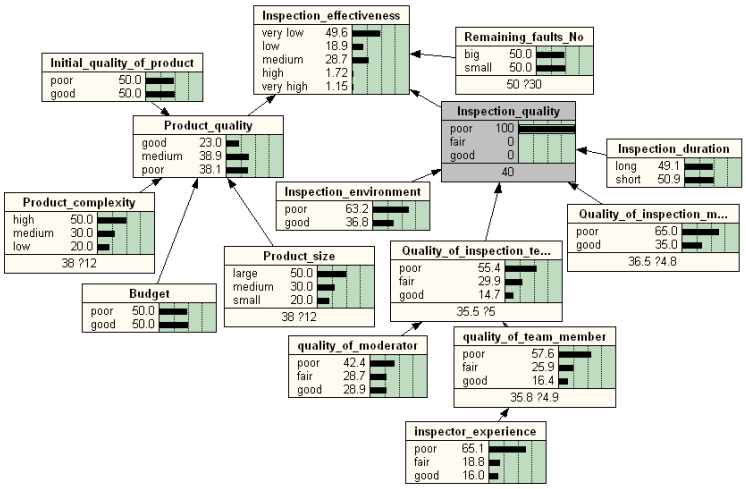


(1)

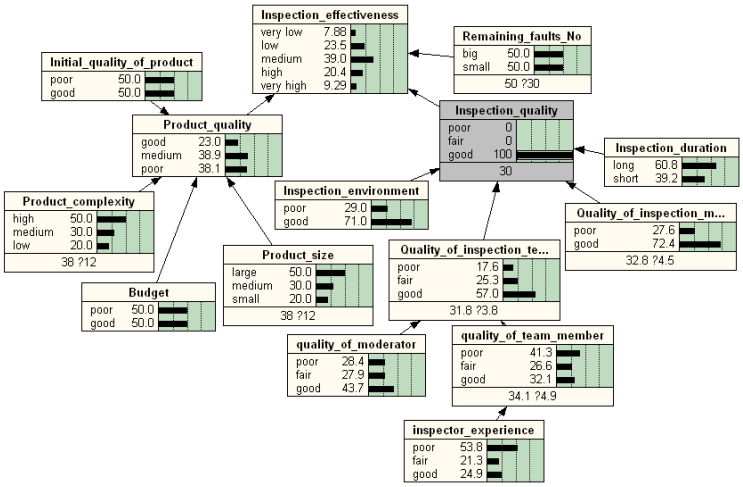


(2)

Figure 7. 7 Change of the probability of product complexity

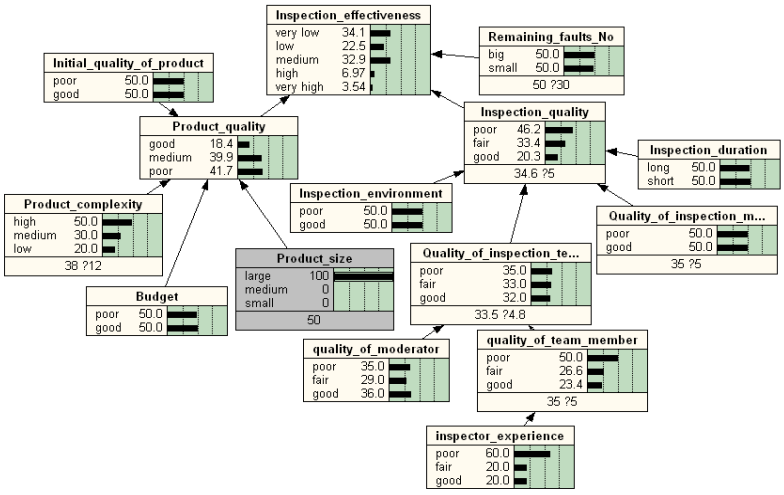


(1)

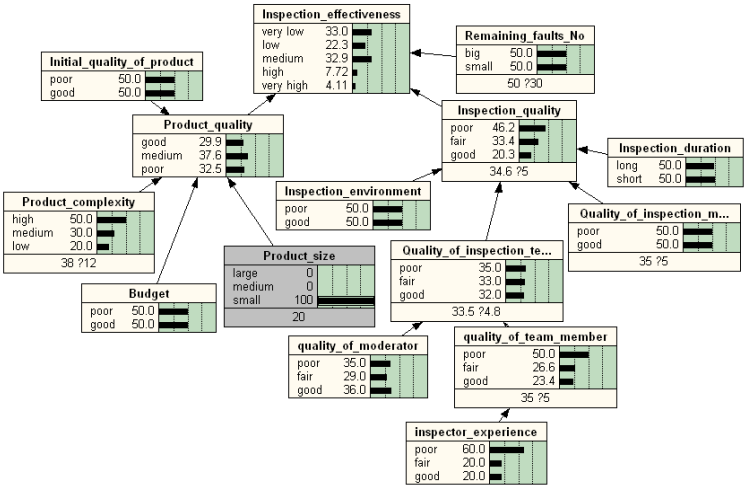


(2)

Figure 7. 8 Change of the probability of quality of inspection process



(1)



(2)

Figure 7. 9 Change of the probability of product size

In addition, entropy reduction, also called mutual information, is used as the criterion here to identify the key attributes of software inspection which help increasing the quality of software. The entropy of a discrete random variable X is defined to be $H(X) = -\sum_{i=1}^{n} \log_2 P(X = x_i) P(X = x_i)$ where P is the probability distribution of X. The

entropy of two discrete random variables X and Y (considered jointly) is given by $H(X,Y) = -\sum_{i=1}^{n} \log_2 P(X = x_i, Y = y_i) P(X = x_i, Y = y_i)$ with $H(X,Y) = H(Y,X)$. The mutual information of X and Y is given by $I(X,Y) = H(X) + H(Y) - H(X,Y)$. It follows from this definition that $I(X,Y) = I(Y,X)$. The mutual information of two random variables is a measure of how much information a knowledge of one of the random variable provides about the other. Entropy reduction is conducted in order to evaluate the degree of heterogeneity or homogeneity of spatial natural resources. It would show the degree of uncertainty represented in the model before and after entering the evidence. That means the higher the entropy reduction is, the more sensitive the node is. With the example shown in the figures above, the related sensitivity analysis results with entropy reduction are shown as Table 7.4, in which the "remaining number of faults" has the most influence over the 'inspection effectiveness', next is the node "inspection quality", the node "product quality" has the least influence over the "inspection effectiveness".

Table 7. 4 Sensitivity analysis with entropy reduction

Important attributes	Entropy reduction	Percentile (%)
Remaining faults No.	0.494	49.40
Inspection quality	0.135	8.95
Product quality	0.021	1.32

In addition, we also interested in how the change of the node "inspection experience" can influence the overall inspection effectiveness. As generally there is a learning process during the inspection, the belief of inspection experience extracted from experts would

show an increasing trend over the inspection process. As a result, the inspection effectiveness should also be evaluated in a dynamic way: updated with each new collected belief over the inspection experience. Such a scheme can take the advantage of using Bayesian Network, and give online feedback and update as the process running. Using our numerical example, we can consider the influence of the change of this node as shown in Table 7.5.

Table 7. 5 Sensitivity analysis of "inspector's experience" with Entropy

Important attributes	Entropy reduction	Percentile (%)
Inspector's experience	0.001304	0.0951

Here a systematic approach is proposed using Bayesian networks to analyze the inspection process. As our numerical example is just for illustrative purpose, more experimental data are needed to give further insight into the inspection process.

7.5 Summary

The above results showed that using BN model could help measuring the inspection effectiveness, and finding out key factors of great influence. Based on that, software manager can carry out some action to improve the effectiveness of inspection. Faults are removed as early as possible, as much as possible, thus the debugging cost at the later stage of testing are reduced and software quality improved as well.

Bayesian networks provide a convenient framework to model the inspection effectiveness with both abstract knowledge and actual data. However, it is not an easy task for belief elicitation over the probability distributions. Stepping from the early work in using Bayesian network to model software inspection process (Cockram, 2001), some further explorations are investigated under two major points. Firstly, we attempt to incorporate the variable of remaining number of faults into the network structure, because this information is nature to influence the belief on inspection effectiveness sequentially and it can be available through estimation or expert opinion. Secondly, a more systematic approach to gather the probability distributions is proposed. Specially, the mathematics-based AHP is introduced to gather the probability distribution. Thirdly, the dynamic analysis on the inspection effectiveness is developed with the Bayesian network, and continuous evaluation on this measure is available to aid related decision-making. With the established method, a numerical example is illustrated to show some applications of these techniques and sensitivity analysis is developed. However, still there are some works for further investigation. AHP can provide more information actually, such as the overall weights for each factor. As a result, it would be interesting to study the application of this information in Bayesian network modeling. Also, the example analysis is not complete and it is favorable to develop systematic analysis with more experimental data for further insight into the inspection process.

Chapter 8 Conclusion and Future Work

The main focus of the work presented in this thesis was to extend the traditional software reliability models through different perspectives and to study the corresponding decision-making problems. This chapter summarizes the results of the research work and discusses their limitations and implications. Recommendations on further research and practical application are also presented.

8.1 Research Results

Software testing process is composed of fault detection, correction, and possible introduction. A major part of the study in this thesis was to incorporate the software fault correction process into software reliability modeling frameworks, relaxing the restrictive assumptions in traditional software reliability models. The models were developed through both analytical and data-driven approaches.

At first, extensions on analytical NHPP software reliability models are presented in chapter 3. A paired FDP and FCP modeling framework is proposed, by assuming the relationship between FDP and FCP is the time delay. Generally, modeling both fault detection and correction processes will provide more information than traditional models. It is more realistic compared with traditional software reliability models as this proposed model takes into account of the time delay.

Further extensions were also carried out within this framework in chapter 4 to obtain the ML estimators of the model parameters. The ML estimated model parameters can give a more accurate estimation of the combined software fault detection and correction process.

In chapter 5 the prediction performance is further analyzed. Experimental results of the simulation analysis show that the ML estimates with a fairly accurate prediction capability compared with the LS estimates. The approach in our study can be further extended to general SRGMs considering the fault detection and correction process.

The corresponding decision-making problems of optimal software release time are further discussed in chapter 6. Many assumptions are relaxed in this cost model, fault debugging time is considered and the warranty and risk cost issues are included. The proposed new economic model can provide more accurate results such as when the mission time being changed, or the warranty period shortened or prolonged.

Besides the analytical approach, this thesis also explored the Bayesian networks applications in the field of software reliability modeling and analysis. As except for software testing, another way to reduce the software faults is through review and walk-through during the inspection process. In chapter 7, Bayesian networks were applied in modeling the software inspection process. Accordingly, this could adaptively update the effectiveness evaluation with new data collected, which could be useful for inspection stopping time determination. Also, a systematic approach to extract the distributions was given, which ensures the feasibility of the application of this kind of model.

8.2 Future Research

Different software is developed under different environment, and the software testing process is influenced by many uncertain factors. As a result, it is difficult to find a universal software reliability model to suit all software testing processes. However, extensions to current software reliability models have been developed by relaxing current restrictive assumptions through incorporating more practical information. Another future topic is as discussed earlier that while the ML estimate of the failure rate of the G-O model was consistent; the ML estimate of parameter *a* of the G-O model was not consistent when the observation period extends to infinity. This could be further analyzed in future research.

Beyond the studies we explored in our current work, some other approaches can be studied in the further. Although analytical NHPP models provided a simple approach for software reliability analysis and release time determination, they were based on a simplified assumption on the relationship between fault detection and correction. This assumption can fit some testing environments where there is little on fault detection from fault correction, but actually slow fault correction could delay fault detection and fast fault correction could add pressure on fault detection. Therefore, the 'feedback' effect from fault correction should be incorporated into the modeling framework. However, the information provided with one-step prediction is quite limited. Multi-step prediction should be carried out to provide more information useful for practical decision-making, as the final goal of software reliability modeling is to help making decision. For both two kinds of models, only one dataset with a few data points are applied in our current case

study. To justify the proposed models, more datasets should be used for applications of the proposed models. Limited by the availability of published dataset, simulation could be an alternative approach to acquire the data.

Furthermore, as software testing process is influenced by many uncertain factors, such as imperfect debugging, change-point, more realistic models can be proposed (Zou, 2003; Xie, et al., 2004b; Park et al., 2005), in addition, it would be interesting to extend this general model in a stochastic way. Some extensions have been done to model the fault detection process with a SDE stochastic differential equation (Yamada et al., 1995; Lee, 2004). However, there are no extensions on fault correction. As an extension to these SDE models, random factors in both fault-detection and correction could be incorporated. Technically, linear stochastic differential equations assure the existence of a unique solution, and it is convenient to consider time-independent conditions. Accordingly, the parameters in the model can be estimated through Maximum Likelihood methods and useful measures are expected to be derived with the model to assist software testing decision making.

At last, some BN models have been proposed dealing with software reliability issues (Fenton and Neil, 1999), and there is still much scope for extending the methods and the applications to reliability problems. The Bayesian Network modeling with software reliability prediction is an interesting topic worth further exploration. Modern mature software companies have many failure datasets within their own database. The flexibility of BN modeling framework provides an approach to utilize this kind of information to

improve the software reliability prediction performance. There is no doubt that BBNs can provide a powerful tool for reasoning with uncertainty.

Answers to these questions will provide more practical modeling and analysis approach for a mature software company. Stepping from our current study on fault detection and correction process modeling, above are some works that can still be left to be covered in the future.

REFERENCES

Amasaki, S., Yoshitomi, T., Mizuno, O., Takagi, Y. and Kikuno, T., 2005, 'A new challenge for applying time series metrics data to software quality estimation', *Software Quality Journal*, vol. 13, no. 2, pp. 177-193.

Aurum, A., Petersson, H. and Wohlin, C. 2002, 'State-of-the-art: software inspections after 25 years', *Software Testing Verification and Reliability*, vol. 12, no. 3, pp. 133-154.

Aurum, A., Wohlin, C., Petersson, H., 2005, 'Increasing the understanding of effectiveness in software inspections using published data sets', *Journal of Research and Practice in Information Technology*, vol. 37, no. 3, pp. 253-266.

Bai and Yun, 1988, 'Optimum number of errors corrected before releasing a software system', *IEEE Transactions on Reliability*, vol. 37, pp. 41-44.

Bazaraa, M. S., Sherali, H. D. and Shetty, C. M., 1993, *Nonlinear programming: theory and algorithms*. John-Wiley and Sons.

Berman, O. and Cutler, M., 2004, 'Resource allocation during tests for optimally reliable software', *Computers and Operations Research*, vol. 31, pp. 1847-1865.

Biffl, S., 2003, 'Evaluating defect estimation models with major defects', *Journal of Systems and Software*, vol. 65, no. 1, pp. 13-29.

Biffl, S. and Halling, M., 2003, 'Investigating the defect detection effectiveness, and cost benefit of nominal inspect-ion teams', *IEEE Transactions on Software Engineering*, vol. 29, no. 5, pp. 385-397.

Boland, P. J. and Chuiv, N. N., 2007, 'Optimal times for software release when repair is imperfect', *Statistics and Probability Letters*, vol. 77, no. 12, pp. 1176-1184.

Briand, L. C., Freimut, B. and Vollei, F., 2004, 'Using multiple adaptive regression splines to support decision making in code inspections', *Journal of Systems and Software*, vol. 73, no. 2, pp. 205-217.

Bustamantea, A. S. and Bustamante, B. S., 2003, 'Multinomial-exponential reliability function: a software reliability model', *Reliability Engineering and System Safety*, vol. 79, pp. 281–288.

Catuneanu, V. M., Moldovan, C., Popentiu, F. L. and Popovici, D., 1991, 'Software reliability release policy with testing effort', *Microelectronic Reliability*, vol. 31, no.5, pp. 895-899.

Chang, Y. C., Hung, W. L., 2005, 'Software release policies on a shot-noise process model'. *Applied Mathematics and Computation*, vol. 171, no. 2, pp.746-759.

Chari, K. and Hevner, A., 2006, 'System test planning of software: an optimization approach', *IEEE Transactions on Software Engineering*, vol. 32, no. 7, pp. 503-509.

Chatterjee, S., Misra, R. B. and Alam, S. S., 2004, 'N-version programming with imperfect debugging', *Computer and Electrical Engineering*, vol. 30, no. 6, pp. 453-463.

Chiu, K. C., Huang, Y.S., Lee, T. Z., 2008, 'A study of software reliability growth from the perspective of learning effects', *Reliability Engineering & System Safety*, vol. 93, no.10, pp. 1410-1421.

Cockram, T., 2001, 'Gaining confidence in software inspection using a Bayesian belief model', *Software Quality Journal,* vol. 9, no. 1, pp. 31-42.

Collofello, J. and Woodfield, S., 1989, 'Evaluating the effectiveness of reliability-assurance techniques', *Journal of Systems and Software*, vol. 9, no. 3, pp. 191-195.

Dai, Y. S., Xie, M., Poh, K. L., 2004, 'A model for correlated failures in N-version programming', *IIE Transactions*, vol. 36, no. 12, pp. 1183-1192.

Dai, Y. S., Xie, M., Poh, K. L., 2005, 'Modeling and analysis of correlated software failures of multiple types', *IEEE Transactions on Reliability*, vol. 54, no. 1, pp. 100-106.

Delic, A. K., Mazzanti, F., Strigini, L, 1995, 'Formalizing a Software Safety Case via Belief Networks', *Technical report*, Center for Software Reliability, City University, London, U.K.

Dohi, T., Teraoka, Y. and Osaki, S., 2000, 'Software release games', *Journal of Optimization Theory and Applications*, vol. 105, no.2, pp. 325-346.

Dohi, T., Nishio, Y. and Osaki, S. 1999, 'Optimal software release scheduling based on artificial neural networks', *Annals of Software Engineering*, vol. 8, pp. 167-185.

Emam, K. E. and Laitenberger, O. and Harbich T., 2000, 'The application of subjective estimates of effectiveness to controlling software inspections', *Journal of Systems and Software*, vol. 54, pp. 119-136.

Emam, K. M. and Laitenberger, O., 2001, 'Evaluating capture-recapture models with two inspectors', *IEEE Transactions on Software Engineering*, vol. 27, no. 9, pp. 851-864.

Expert Choice, 2005, http://www.expertchoice.com/

Fagan, M. 1976, 'Design and code inspections to reduce errors in program development', *IBM Systems Journal,* no.3, pp. 219 – 248.

Fagan, M. 1986, 'Advances in software inspections', *IEEE Transactions on Software Engineering*, vol. 12, no. 7, pp. 744-751.

Fan, C. F. and Yu, Y. C., 2004, 'BBN-based software project risk management', *Journal of Systems and Software,* vol. 73, no. 2, pp. 193-203.

Fenton, N. and Neil, M., 1999, 'Software metrics: successes, failures and new directions', *Journal of Systems and Software,* vol. 47, pp. 149-157.

Franz, L. A. and Shih, J. C., 1994, 'Estimating the value of inspections and early testing for software projects', *CS-TR-6, Hewlett-Packard Journal*, pp. 60-67.

Freimut, B., Briand, L. C., and Vollei, F., 2005, 'Determining inspection cost-effectiveness by combining project data and expert opinion', *IEEE Transactions on Software Engineering*, vol. 31, no. 12, pp. 1074-1092.

Gilb, T. and Graham, D., 1993, *Software Inspection*, Addison-Wesley.

Grady, R. and Slack, T. V., 1994, 'Key lessons in achieving widespread inspection use', *IEEE Software*, vol. 11, no. 4, pp. 46-57.

Goel, A. L. and Okumoto, K., 1979, 'Time-dependent error-detection rate model for software reliability and other performance measures', *IEEE Transactions on Reliability*, vol. 28, pp. 206-211.

Gokhale, S. S. 2003, 'Optimal software release time incorporating fault correction', *Proceedings of the 28th Annual NASA Goddard Software Engineering Workshop (SEW'03),* pp. 175-184.

Gokhale, S. S., Lyu M. R. and Trivedi, K. S. 2006, 'Incorporating fault debugging activities into software reliability models: a simulation approach', *IEEE Transactions on Reliability*, vol. 55, no. 2, pp. 281-292.

Hou, R. H., Kuo, S. Y. and Chang, Y. P., 1997, 'Optimal release times for software systems with scheduled delivery time-based on the HGMM', *IEEE Transactions on Computer*, vol. 46, pp. 216-221.

Hu, Q. P. , Xie, M., Ng, S. H. and Levintin, G., 2007, 'Robust recurrent neural network modeling for software fault detection and correction prediction', *Reliability Engineering and System Safety*, vol. 92, no. 3, pp. 332-340.

Huang, C. Y., Lyu, M. R. and Kuo, S. Y., 2003, 'A unified scheme of some NH PP models for software reliability estimation'. *IEEE Transactions on Software Engineering,* vol. 29, no. 3, pp. 261-269.

Huang, C. Y. 2005a, 'Cost-reliability-optimal policy for software reliability models incorporating improvements in testing efficiency', *Journal of Systems and Software*, vol. 77, pp. 139-155.

Huang, C. Y. 2005b, 'Performance analysis of software reliability growth models with testing-effort and change-point', *Journal of Systems and Software,* vol. 76, pp.181-194.

Huang, C.Y. and Lyu, M. R., 2005a, 'Optimal release time for software systems considering cost, testing-effort, and test efficiency', *IEEE Transactions on Reliability*, vol. 54, no. 4, pp. 583-591.

Huang, C. Y. and Lyu, M. R., 2005b, 'Optimal testing resource allocation, and sensitivity analysis in software development', *IEEE Transactions on Reliability*, vol. 54, no. 4, pp.592-603.

Huang, C.Y. and Lin, C. D., 2006, 'Software reliability analysis by considering fault dependency and debugging time lag', *IEEE Transactions on Reliability*, vol. 55, no. 3, pp. 436-450.

Huang, C. Y. and Lo, J. H., 2006, 'Optimal resource allocation for cost and reliability of modular software systems in the testing phase', *Journal of Systems and Software*, vol. 79, pp. 653-664.

Huang, C. Y., Kuo, S. Y. and Lyu, M. R., 2007, 'An assessment of testing-effort dependent software reliability growth models', *IEEE Transactions on Reliability*, vol. 56, no. 2, pp. 198-211.

Inoue, S. and Yamada, S., 2004, 'Testing-coverage dependent software reliability growth modeling', *International Journal of Reliability, Quality and Safety Engineering,* vol. 11, no. 4, pp. 303-312.

Inoue, S. and Yamada, S., 2006, 'Discrete software reliability assessment with discretized NHPP models', *Computers and Mathematics with Applications*, vol. 51, no. 2, pp. 161-170.

Jain, M. and Maheshwari, S., 2006, 'Generalized renewal process (GRP) for the analysis of software reliability growth model', *Asia-Pacific Journal of Operational Research*, vol. 23, no. 2, pp. 215-227.

Jeske, D.R. and Pham, H., 2001, 'On the maximum likelihood estimates for the Goel--Okumoto software reliability model', *The American Statistician*, vol. 55, no. 3, pp. 219-222.

Jiang, L. T. and Xu, G. Z., 2007, 'Modeling and analysis of software aging and software failure', *Journal of Systems and Software*, vol. 80, no. 4, pp. 590-595.

Kapur, P. K., Garg, R. B. and Bhalla, V. K. 1993, 'Release policies with random software life cycle and penalty cost', *Microelectronic Reliability*, vol. 33, no.1, pp. 7-12.

Kapur, P. K., Goswami, D. N., Bardhan, A. and Singh, O., 2008, 'Flexible software reliability growth model with testing effort dependent learning process', *Applied Mathematical Modeling*, vol. 32, no. 7, pp.1298-1307.

Karunanithi, N., Whitley, D. K. and Malaiya, Y., 1992, 'Using neural networks in reliability prediction', *IEEE Transactions on Software Engineering*, pp. 53-59.

Kelly, D. and Shepard, T., 2004a, 'Eight maxims for software inspectors', *Software Testing Verification and Reliability,* vol. 14, no. 4, pp. 243-256.

Kelly, D. and Shepard, T., 2004b, 'Task-directed software inspection', *Journal of Systems and Software*, vol. 73, no. 2, pp. 361-368.

Keefer, D. L. and Bodily, S. E., 1983, 'Three-point approximations for continuous random variables', *Management Science,* vol. 29, no. 5, pp. 595-609.

Khoshgoftaar, T. M., 1988, 'Non-homogenous Poisson Processes for software reliability growth', *Proceedings of 8th Symposium in Computational Statistics*, pp. 11-12.

Kimura, M., Toyota, T. and Yamada, S., 1999, 'Economic analysis of software release problems with warranty cost and reliability requirement', *Reliability Engineering and System Safety,* vol. 66, pp. 49-55.

Kollanus, S., 2005, 'Issues in software inspection practices', *Lecture Notes in Computer Science,* vol., 3547, pp. 429-442.

Kusumoto, S., Matsumoto, K., Kikuno, T. and Torii, K., 1992, 'A new metric for cost-effectiveness of software reviews', *IEICE transactions on Information and Systems*, vol. E75-D, no. 5, pp. 674-680.

Laitenberger, O., Baud J. M., 2000, 'An encompassing life cycle centric survey of software inspection', *Journal of Systems and Software*, vol. 50, pp. 5-31.

Leung, Y. W., 1992, 'Optimum software release time with a given cost budget', *Journal of Systems and Software*, vol. 17, pp. 233-242.

Levitin, G., 2005, 'Optimal structure of fault-tolerant software systems', *Reliability Engineering and System Safety*, vol. 89, no. 3, pp. 286-295.

Levitin, G., Xie, M., 2006, 'Performance distribution of a fault-tolerant system in the presence of failure correlation', *IIE Transactions*, vol. 38, no. 6, pp. 499-509.

Levitin, G., Xie, M., Zhang, T. L., 2007, 'Reliability of fault-tolerant systems with parallel task processing', *European Journal of Operational Research*, vol. 177, no. 1, pp. 420-430.

Li, S. M., Yin, Q., Guo, P. and Lyu, M. R., 2007, 'A hierarchical mixture model for software reliability prediction', *Applied Mathematics and Computation*, vol. 185, pp. 1120 – 1130.

Lin, C.T. and Huang, C. Y., 2008, 'Enhancing and measuring the predictive capabilities of testing-effort dependent software reliability models', *J. of Systems and Software*, vol. 81, no. 6, pp.1025-1038.

Lyu, M. R. 1996, *Handbook of Software Reliability Engineering,* McGraw-Hill, New York.

McDaid, K. and Wilson, S. P., 1997, 'Optimal software testing under a time dependent error detection rate model', *Technical Report*, Department of statistics, Trinity College, Dublin, Ireland, 1997.

McDaid, K. and Wilson, S. P., 2001, 'Deciding how long to test software', *Statistician*, vol. 50, pp. 117-134.

Miller, J. and Yin, Z. C., 2004, 'A cognitive-based mechanism for constructing software inspection teams', *IEEE Transactions on Software Engineering*, vol. 30, no. 11, pp. 811-825.

Morali, N. and Soyer, R., 2003, 'Optimal stopping in software testing', *Naval Research Logistics*, vol. 50, pp.88-104.

Musa, J. D., 1993, 'Operational profiles in software reliability engineering', *IEEE Software*, vol. 10, pp. 14-32.

Musa, J. D., Iannino, A. and Okumoto, K., 1987, *Software reliability: measurement, prediction, application,* McGraw-Hill, New York.

Myrtveit, I., Stensrud, E. and Shepperd, M., 2005, 'Reliability and validity in comparative studies of software prediction models', *IEEE Transactions on Software Engineering*, vol. 31, no. 5, pp. 380-391.

Nayak, K. T., Bose, S. and Kundu, S., 2008, 'On inconsistency of estimators of parameters of non-homogeneous Poisson process models for software reliability', *Statistics & Probability Letters*, vol. 78, no. 14, pp. 2217 – 2221.

NETICA, 2005, http://www.norsys.com/netica.html

Nishio, Y. and Dohi, T., 2003, 'Determination of the optimal software release time based on proportional hazards software reliability growth models', *Journal of Quality in Maintenance Engineering*, vol. 9, no. 1, pp. 48-65.

Nishiwaki, M., Yamada, S. and Ichimori, T., 1996, 'Testing resource allocation policies based on an optimal software release problem', *Mathematica Japonica*, vol. 43, no. 1, pp. 91-97.

Ohtera, H., Yamada, S., 1990, 'Optimum software-release time considering an error-detection phenomenon during operation', *IEEE Transactions on Reliability*, vol. 39, no.5.

Okumoto, K. and Goel, A. L., 1980, 'Optimum release time for software systems based on reliability and cost criteria', *Journal of Systems and Software*, vol. 1, pp. 315-318.

Ozekici, S. and Catkan, 1993, 'A dynamic software release model', *Computer and Economics*, Vol. 6, pp. 77-94.

Ozekici, S., Altinel, K. and Ozcelikyurek, S., 2000, 'Testing of software with an operational profile', *Naval Research Logistics*, vol. 47, no. 8, pp. 620-634.

Park, J. Y., Hwang, Y. S., and Fujiwara, T., 2005, 'Integration of imperfect debugging in general testing-domain dependent NHPP SRGM', *International Jjournal of Reliability, Quality and Safety Engineering*, vol. 12, no. 6, pp. 493-505.

Pearl, J., 1986, 'Fusion, propagation and structuring in belief network', *Artificial Intelligence*, vol. 29, pp. 241-288

Perry, D. E., Porter, A., Wade, M. W., Votta, L. G. and Perpich, J., 2002, 'Reducing inspection interval in large-scale software development', *IEEE Transactions on Software Engineering,* vol. 28, no. 7, pp. 695-705.

Petersson, H., Thelin, T., Runeson, P. and Wohlin, C., 2004, 'Capture-recapture in software inspections after 10 years research - theory, evaluation and application', *Journal of Systems and Software*, vol. 72, no. 2, pp. 249-264.

Petrova, E. and Malevris, N., 1992, 'Rules and criteria for when to stop testing a piece of software', *Microelectronic Reliability,* vol. 32, no. ½, pp. 101-117.

Pham, H., 1996, 'A software cost model with imperfect debugging, random life cycle and penalty cost', *International Journal of Systems Science*, vol. 27, no. 5, pp. 455-463.

Pham, H., 2000, *Software Reliability*, Springer-Verlag, New York.

Pham, H., 2003, 'Software reliability and cost models: perspectives, comparison, and practice', *European Journal of Operational Research*, vol. 149, pp. 475-489.

Pham, H. and Zhang, X.M., 1999 a, 'A software cost model with warranty and risk costs', *IEEE Transactions on Computers*, vol. 48, no. 1, pp. 71-75.

Pham, H. and Zhang, X. M., 1999 b, 'Software release policies with gain in reliability justifying the costs', *Annals of Software Engineering*, no. 8, pp. 147-166.

Pham, H., Zhang, X. M., 2003, 'NHPP software reliability and cost models with testing coverage', *European journal of Operational Research*, vol. 145, pp. 433-454.

Pham, H. and Wang, H. Z., 2001, 'A quasi-renewal process for software reliability and testing costs', *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans,* vol. 31, no. 6, pp. 623-631.

Pham, L. and Pham, H., 2000, 'Software reliability models with time-dependent hazard function based on Bayesian approach', *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 30, no. 1, pp. 25-35.

Pham, L. and Pham, H., 2001, 'A Bayesian predictive software reliability model with pseudo-failures', *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans,* vol. 31, no. 3, pp.233-238.

Porter, A. A., Siy, H. P., Toman, C. A. and Votta, L. G., 1997, 'An experiment to assess the cost-benefits of code inspections in large scale software development', *IEEE Transactions on Software Engineering*, vol. 23, no. 6, pp. 329-346.

Remus, H. and Zilles, S., 1979, 'Prediction and management of program quality', *4th National Conference on Software Engineering, IEEE Computer Society Press*, pp. 341-350.

Rico, D.F., 2004, *ROI of software process improvement*, J. Ross Publishing.

Rinsaka, K. and Dohi, T., 2004, 'Who solved the optimal software release problems based on Markovian software reliability model?', *The 47th IEEE International Midwest Symposium on Circuits and Systems,* pp. 475-478.

Rosqvist, T., Koskela, M. and Harju, H., 2003, 'Software quality evaluation based on expert judgment', *Software Quality Journal,* vol. 11, pp. 39-55.

Saaty, T. L., 1980, *The analytic hierarchy process: planning, priority setting, resource allocation*, McGraw-Hill.

Schneidewind, N. F., 1975, 'Analysis of error processes in computer software', *Proceedings of International Conference on Reliable Software, IEEE Computer Society,* pp. 337-346.

Schneidewind, N. F., 1993, 'Software reliability model with optimal selection of failure data', *IEEE Transactions on Software Engineering*, vol. 19, no. 11, pp. 1095-1104.

Schneidewind, N. F., 2001, 'Modeling the fault correction process', *Proceedings of the 12th International Symposium on Software Reliability Engineering,* pp. 185-190.

Schneidewind, N. F., 2003, 'Fault correction profiles', *Proceedings of the 14th International Symposium on Software Reliability Engineering,* pp. 257-267.

Schneidewind, N. F., 2005, 'Predicting risk as a function of risk factors', *Innovations Systems Engineering*, no. 1, pp. 63-70.

Sharma, V. S. and Trivedi, K. S., 2007, 'Quantifying software performance, reliability and security: an architecture-based approach', *Journal of Systems and Software*, vol. 80, pp. 493-509.

Shyur, H. J., 2003, 'A stochastic software reliability model with imperfect-debugging and change-point', *Journal of Systems and Software*, vol. 66, no. 2, pp. 135-141.

Smidts, C., Stoddard, R.W. and Stutzke, M., 1998, 'Software reliability models: an approach to early reliability prediction', *IEEE Transactions on Reliability*, vol. 47, pp. 268- 278.

Stringfellow, C., Andrews, A., Wohlin, C. and Petersson, H., 2002, 'Estimating the number of components with defects post-release that showed no defects in testing', *Software Testing, Verification and Reliability,* vol. 12, pp. 93-122.

Stutzke, M. A. and Smidts, C. S., 2001, 'A stochastic model of fault introduction and removal during software development', *IEEE Transactions on Reliability*, vol. 50, pp.184-193.

Tamura, Y. and Yamada, S., 2006, 'A flexible stochastic differential equation model in distributed development environment', *European Journal of Operational Research*. vol. 168, no. 1, pp. 143-152.

Tausworthe, R. C. and Lyu, M. R., 1996a, 'A Generalized Technique for Simulating Software Reliability', *IEEE Software*, vol. 13, no. 2, pp. 77-88.

Tausworthe, R. C. and Lyu, M. R., 1996b, 'Handbook of Software Reliability Engineering', *Chapter Software Reliability Simulation*, pp. 661-698, *McGraw-Hill, New York.*

Teng, X. L. and Pham, H., 2004, 'A software cost model for quantifying the gain with considerations of random field environments', *IEEE Transactions on Computers*, vol. 53, no.3, pp. 380-384.

Teng, X. L. and Pham, H., 2006, 'A new methodology for predicting software reliability in the random field environments', *IEEE Transactions on Reliability*, vol. 55, no. 3, pp. 458-468.

Wood, A. P., 1996, 'Predicting software reliability', *IEEE Computer*, pp. 69-77.

Wu, Y. P., Hu, Q. P., Xie, M. and Ng, S. H., 2007, 'Modeling and analysis of software fault detection and correction process by considering time dependency', *IEEE Transactions on Reliability*, vol. 56, no. 4, pp.629-642.

Xia, G. L., Zeephongsekul, P. and Kumar, S., 1993, 'Optimal software release policy with a learning factor for imperfect debugging', *Microelectronic Reliability*, vol. 33, no. 1, pp. 81-86.

Xie, M., 1991, *Software reliability modeling,* World Scientific, Singapore.

Xie, M. and Zhao, M., 1992, 'The Schneidewind software reliability model revisited', *Proceedings of the 3rd International Symposium on Software Reliability Engineering,* pp. 184-192.

Xie, M. and Hong, G. Y., 1998, 'A study of the sensitivity of software release time', *Journal of Systems and Software,* vol. 44, pp. 163-168.

Xie, M. and Hong, G. Y., 1999, 'Software release time determination based on unbounded NHPP model', *Computers and Industrial Engineering,* vol. 37, pp. 165-168.

Xie, M., Hong, G. Y. and Wohlin, C., 1999, 'Software reliability prediction incorporating information from a similar project', *Journal of Systems and Software,* vol. 49, no. 1, pp. 43-48.

Xie, M. and Yang, B., 2003, 'A study of the effect of imperfect debugging on software development cost', *IEEE Transactions on Software Engineering*, vol. 29, no. 5, pp. 471-473.

Xie, M., Dai, Y. S., Poh, K. L., Lai, C. D., 2004a. 'Optimal number of hosts in a distributed system based on cost criteria', *International Journal of Systems Science*, vol. 35, no. 6, pp. 343-353.

Xie, M, Dai, Y. S., Poh, K. L., Lai, C. D., 2004b, 'Distributed system availability in the case of imperfect debugging process', *International Journal of Industrial Engineering-Theory Applications and Practice,* vol. 11, no. 4, pp. 396-405.

Xie, M., Hu, Q. P., Wu, Y. P., Ng, S. H., 2007, 'A study of the modeling and analysis of software fault-detection and fault-correction processes', *Quality and Reliability Engineering International,* vol. 23, no. 4, pp. 459-470.

Yamada, S., Ohba, M. and Osaki, S., 1984a, 'S-shaped software reliability growth models and their applications', *IEEE Transactions on Reliability*, vol. R-33, no. 4, pp. 289-292.

Yamada, S., Narihisa, and Osaki, S., 1984b, 'Optimal release policies for a software system with scheduled software delivery times', *International Journal of Systems Science,* vol. 15, pp. 905-914.

Yamada, S., Ichimori, T. and Nishiwaki, M., 1995, 'Optimal allocation policies for testing resource based on a software reliability growth model', *International Journal of Mathematical and Computer Modeling,* vol. 22, no. 10-12, pp. 295-301.

Yang, M. and Chao, A. 1995, 'Reliability estimation and stopping rules for software testing, based on repeated appearance of bugs', *IEEE Transactions on Reliability*, vol. 44, no. 2, pp. 315-321.

Yang, B. and Xie, M., 2000, 'A study of operational and testing reliability in software reliability analysis', *Reliability Engineering and System Safety*, vol. 70, pp. 323-329.

Yin, Z. C., Dunsmore, A. and Miller, J., 2004, 'Self-assessment of performance in software inspection processes', *Information and Software Technology*, vol. 46, no. 3, pp. 185-194.

Zhang, X. M. and Pham, H., 1998, 'A software cost model with warranty cost, error removal times and risk costs', *IIE Transactions*, vol. 30, pp. 1135-1142.

Zhang, X. M., Teng, X. L. and Pham, H., 2003, 'Considering fault removal efficiency in software reliability assessment', *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans,* vol. 33, no. 1, pp. 114-120.

Zhao, J. , Liu, H. W., Cui, G. and Yang, X. Z., 2006, 'Software reliability growth model with change-point and environmental function', *Journal of Systems and Software*, vol. 79, no. 11, pp. 1578-1587.

Zou, F. Z., 2003, 'A change-point perspective on the software failure process', *Software Testing, Verification and Reliability*, vol. 13, pp. 85-93.